The Resource Journal For IBM PC Programmers

PROGRAMMER'S Journal

JANUARY/FEBRUARY 1987
VOLUME 5.1
$3.95

ADOS COMETH

The New DOS—Programming in the Protected Mode
by Frank D. Greco

Michael Abrash—
Inside The EGA

Tom Swan—
How Many Holes Does it Take to Fill an IBM PC?

Bernard Robinson—
Tips on Formatting BASIC Data

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ACTOR WILL DO WINDOWS

If you've been wondering how you're going to explore Microsoft® Windows and then work it into your programs, what you need has just arrived.

ACTOR.™

ACTOR is a new programming system, and there are two reasons why it's so good with Windows.

First, ACTOR is an ideal programming environment. It shows you all your work through Windows. Any part of the program you're writing, its output as it runs, error diagnostics and tracing, catalogs of routines, and, in fact, just about anything you like – they can all be visible on the screen and accessible at the same time, in their own windows.

So it's easier to see what you're doing.

Second, ACTOR is an object-oriented programming language. Everything in an ACTOR program is a self-contained package with its own data and procedures. That is, an object. And the objects pass messages to each other while the program runs.

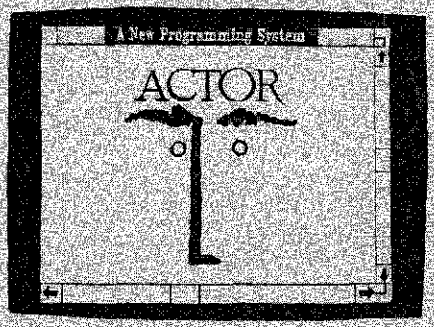
With ACTOR, you don't write lists of instructions. Instead, you define what classes of objects do. Which is especially easy because they inherit most of their behavior from other classes. That means there's less for you to worry about.

And when you create an object, you just choose its name and starting values. After that it pretty much takes care of itself.

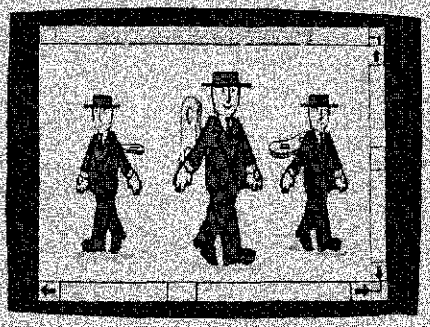
What's important here is that windows are objects like everything else in the language. You make as many as you need, and they do what they're supposed to, mostly on their own.

So it's natural and easy to put windows in your application programs. And users will benefit from Windows when your programs run, as much as you do when you write them.

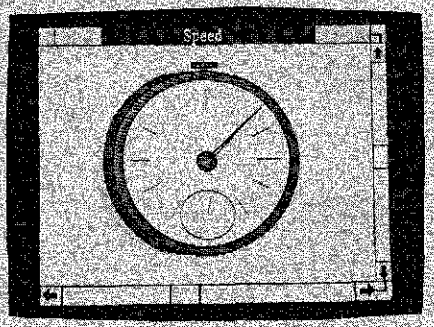
To order, just send in the coupon. Or call **312-491-2370** for more information. You'll see that ACTOR will do Windows, and more.



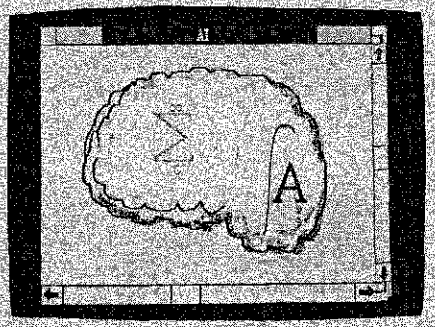
ACTOR™ is a new, interactive programming system, the first with Microsoft® Windows. You have Windows when you write a program, and users have Windows when they run it.



Moreover, ACTOR is a new, object-oriented programming language. Winding objects up and turning them loose is a lot easier, more productive, and more fun than old fashioned programming.



Thanks to a new method of "incremental garbage collection," ACTOR never has to slow down to clean up memory like other advanced languages. You can even use it for real-time control. It's that fast.



ACTOR offers all the features of an ideal artificial intelligence language, but in a familiar Pascal-style syntax. Which makes artificial intelligence programming easier. And what could be smarter than that?

Please send me more information about ACTOR.

Name_____

Address_____

City_____ State_____ Zip_____

Daytime phone_____

☐ Check or money order enclosed. ☐ Bill my VISA. ☐ Bill my MasterCard.

Card number_____ Exp. date_____

☐ I'd like to place an order while I'm at it. (ACTOR is not copy protected and has a 60-day money back guarantee. Educational pricing is available. ACTOR requires an IBM PC/XT or AT or equivalent. 640K and hard disk is recommended.)

☐ ___ copies of ACTOR with Windows runtime support at $495 each. $_____

☐ ___ copies of ACTOR, including Windows complete with MS-DOS Executive, at $550 each. $_____

■ Please add $5 for shipping within the U.S., $30 overseas. $_____

■ Total. $_____

MICROSOFT WINDOWS

The Whitewater Group®

Technology Innovation Center
906 University Place, Evanston, IL 60201

Circle Number 73 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BACK ISSUES

•Vol. 1, No. 1: Norton batch files and Pascal I/O; Program style manual; COBOL on the IBM PC; Benchmarks; System analysis; BASIC modules; Rounding error.

•Vol. 1, No. 2: Defining character sets; C86 BIOS calls; Troubleshooting; Copy protection; Pascal code; Input routines; Writing an assembler.

•Vol. 1, No. 3: ASM dumb terminal; ASM tutorial routines; C source code; ASM graphics line drawing; Text editors reviewed; Utilities; Self-documenting programs; Text files to dBASE II; dBASE II PEEKS & POKES.

•Vol. 1, No. 4: Diskette directory via Pascal; MASM WHILEWEND; C/ASM interface; ASM stay resident routine; Structured programming; Software law; Pascal tutorial; Default values and variable names.

•Vol. 1, No. 5: Multi-tasking software review; Transporting C programs; Programmer's calculator; dBASE II bargraphs; Utilities; ROM BIOS equates; Pascal check program; Combining modules in Assembler; Structured programming in BASIC; Agreement outline.

•Vol. 1, No. 6: ASM printer configuration; DOS patch utility; Hi-speed graphics; Software ownership; Structured programming revisited; C and BASIC compared; Software protection; dWINDOW review.

•Vol. 2, No. 1: Freelance consulting; Sorting in C; Pascal text editor source code; CRT distress; Compiled BASIC; Assembly modules for C; Designing a relational database; Ultra Utilities; Punctuation and Style reviewed.

•Vol. 2, No. 2: Interview with Brian Kernighan; Understanding C through assembly language; Utility routines in C and assembly; Music in C; Automatic line number optimizing in BASIC; Assembly language tutorial; Full-screen input routines in BASIC; dBASE II printer control codes; Pascal tools reviewed; C programming support products.

•Vol. 2, No. 3: Software development environments; Creating a productive screen handling tool; Assembly tutorial; Generating graphics within windows; Composition and editing system for musical ideas; Hybridizing assembly and BASIC; Creating screens; Linking Assembler to Pascal and BASIC.

•Vol. 2, No. 4: Writing a custom ROM BIOS for PC DOS; RISC, Transputers, and mass storage; The future of disk protection; 8088 assembly macros; Random access files in BASIC; C utility routines you can buy; Using the PMATE macro language; Choosing a database manager; E/SP syntax-checking; Pascal editor reviewed.

•Vol. 2, No. 5: Developing the APX CORE Executive; From assembler to C via a 211 byte .EXE main( ); Scroll control using BASIC and Assembler; Faster screen handling with BASIC and ASM; Random access files Part 2; RTOOL, a PC-resident programmer's toolkit; dBkey reviewed; Pascal's pointer power.

•Vol. 2, No. 6: Periscope—building a hardware board for the IBM PC; Intel 80286 bugs by FNE; AT&T surprises all; Field Usage and pointer subtraction in C; Calendar functions for BASIC; Memory segmentation; Reduce your .EXE file sizes; Enhancing Debug; dBASE II to dBASE III? THINKTANK reviewed.

•Vol. 3, No. 1: November's Comdex; C—its uses and abuses; Using the C preprocessor; Memory segmentation examples; Index files, Part 1; Structured programming language; Periscope, a symbolic debugger.

•Vol. 3, No. 2: UNIX! (again); Artificial Intelligence; Doctor C's tips—pointers to functions; Calendar date routines in BASIC; Calculate record layouts, automatically!

SUBSCRIBE and SAVE 17%!

No need to send money— we will bill you later.

Save yourself time, money and energy— have PJ delivered direct to your door every other month.

ONE YEAR (SIX ISSUES)
Reg. ~~$24.00~~
NOW:
$19.95

Name _____

Company _____

Address _____

City _____ State ____ Zip _____

Telephone _____

K 5625

$29.95 Canada / Mexico — $39.95 Foreign PREPAID ONLY U.S. Funds

SUBSCRIBE and SAVE 17%!

No need to send money— we will bill you later.

Save yourself time, money and energy— have PJ delivered direct to your door every other month.

ONE YEAR (SIX ISSUES)
Reg. ~~$24.00~~
NOW:
$19.95

Name _____

Company _____

Address _____

City _____ State ____ Zip _____

Telephone _____

K 5625

$29.95 Canada / Mexico — $39.95 Foreign PREPAID ONLY U.S. Funds

T-SHIRTS & BACK ISSUES

SEE INSIDE BACK COVER FOR AVAILABLE DESIGNS

Please use your street address for UPS shipment. Fill out the back of this card—Thanks

DESIGN NO.	QTY.	COLOR	SIZE	AMOUNT
TOTAL ENCLOSED				

☐ I'll subscribe for a year for $19.95

Back Issues: $5.50 ea PREPAID
$6.50 Can/Mex, $8.50 Foreign
U.S. FUNDS ONLY PLEASE

VOL. 1 _____

VOL. 2 _____

VOL. 3 _____

VOL. 4 _____

REMEMBER— You can have your favorite PJ cartoon made into a custom T-shirt design. Minimum order 2 dozen. Call: (503) 484-2162 for details

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 44 EUGENE, OR

POSTAGE WILL BE PAID BY ADDRESSEE

PROGRAMMER'S JOURNAL
P. O. BOX 3000
Department EE
Denville, NJ 07834

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 44 EUGENE, OR

POSTAGE WILL BE PAID BY ADDRESSEE

PROGRAMMER'S JOURNAL
P. O. BOX 3000
Department EE
Denville, NJ 07834

Please use your street address for UPS shipment

Name _____

Company _____

Address _____

City _____ State ____ Zip _____

Telephone _____

☐ CHECK ENCLOSED

CHARGE MY: ☐ VISA ☐ MC _____
Signature

#_____ Exp. date _____

T-Shirts
and Back Issues
are Prepaid Only—
Please Send
your Check
or Visa/MC Info to:
Programmer's Journal
29 W. 29th Suite 5
Eugene, OR 97405

- Vol. 3, No. 3: NAPLPS and the professional programmer; Instant-C; Printer control from a batch file; Repositioning the stars in Starfinder; Reducing EXE file size through data positioning; tools for better programming.
- Vol 3, No. 4: More UNIX stuff; C programming books, Part 1; C code excerpts, Part 1; How to cure the "dreaded locked keyboard" condition; Calling MS DOS from Turbo Pascal; DOS 2.x sort and find filters; the SET Tools.
- Vol 3, No. 5: Windows done here; C programming books, Part 2; C code excerpts, Part 2; A menu-driven PC using DISPLAY.COM; Turbo Pascal and MASM; making them work together; dBASE II routines; Turbo Pascal 3.0; TopView Programmer's Toolkit.
- Vol. 3, No. 6: About C's HLL overhead; How to recover all your costs; Report on the Comdex and Usenix shows; The Santa Cruz Operation XENIX for the IBM PC; Common C bugs; CTOOLS2; Turbo Pascal 3.0 and subdirectories; More dBASE II routines; Clipper reviewed; A better way to interface BASIC and Assembler.
- Vol. 4, No. 1: FNE on PC/AT hard disk problems and plunging 64K DRAM prices; AUTOINC explained; Sorting using an AVL tree; Combining binary trees; Using DOS to call programs from Turbo Pascal; An operating system for the people; Final installment of Brett Salter's dTOOLS; Two microcomputer graphics books reviewed.
- Vol. 4, No. 2: FNE on Intel 80386, single- vs. multi-user, multi-tasking computing, benchmarks; The home office syndrome; Frank Greco's industry news; Funky string functions; Capturing file handles in Turbo Pascal; Proposed ANSI C Standard, Part 1; Optimizing for speed in Assembler; Memory resident control over print screen functions; Structuring BASIC output with field statements; QuickBASIC Compiler and networking; Programming for Printer Output reviewed.
- Vol. 4, No. 3: FNE—another RISC machine, the paging game, software viewpoint, programmer talent, and Aloha, 52032; Sun Microsystems' next generation PC; Consulting—reusable modules; INT24.PAS, a critical error handler for Turbo Pascal; Memory model independence in Assembler; An approach to windowing for the assembly language programmer; Proposed ANSI C Standard, Part 2; Microsoft and Lattice C 5.0 reviewed; Accelerator boards for software development reviewed.
- Vol. 4, No. 4: Hal W. Hardenbergh on adult toys, tomorrow's micros, HALGOL update, benchmarks, and RISCs; Frank D. Greco on parallel computers, RISC, UNIX; Consulting—efficient custom programming; DOS 2 file handles and file I/O; Programming the WordStar document file conversion utility; More optimizing for speed; Teach an old compiler a new trick; Memory resident programs in C; Tips on formatting with BASIC PRINT USING statements; The structure of dBASE II and III file headers.
- Vol. 4, No. 5: Hal W. Hardenbergh on 32-bit PCs, the 68040, Weitek's data-path part, etc.; Frank D. Greco on large screen monitors, language translators, UNIX, and more; Prolog intro, Turbo Prolog reviewed; Consulting—tax traps, consulting styles, nest eggs; ASM—Search and Display; ASM—the GROUP statement; The proposed ANSI C Standard, Part 3; C—measuring stack usage and Lattice C's startup code; Short boards reviewed.
- Vol. 4, No. 6: DTACK GROUNDED; Unix Books, Part 1; How many amoebas at one o'clock; DOS functions 45H and 46H; DOS Terminate and Stay Resident; Turbo Pascal inline code; Pascal—Assembler souce code for DOS function calls; Alignment problems in C and managing memory allocation; Proposed ANSI C Language Standard, Part 4; The Canon A1 laser printer; Genifer dBASE applications generator reviewed; Advanced Trace86 debugger reviewed.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMPUTER FAIRES® GIVE YOU AN UNFAIR ADVANTAGE.

The Northeast and West Coast Computer Faires are going to help you surge ahead into new business opportunities in 1987. Attendees will find the latest and most innovative products and services to put them ahead in the quest for productivity. Exhibitors searching for a sophisticated audience that's ready to buy will be smiling at their extra sales.

And extensive promotion, a full conference series, and special features like State-of-the-Art Professional Development Seminars help add up to give you the Computer Faire advantage—a smile of triumph.

Attention exhibitors! Call (617) 449-6600, Ext. 4014, now to reserve exhibit space or to receive more information.

West Coast Computer Faire '87
March 26–29, 1987, Moscone Center, San Francisco

Northeast Computer Faire '87
October 15–17, 1987, World Trade Center, Boston

I want the Computer Faire advantage.

Send me ☐ exhibitor ☐ attendee information on ☐ Northeast ☐ West Coast Computer Faires

Name _____ PG1186

Title _____

Company _____

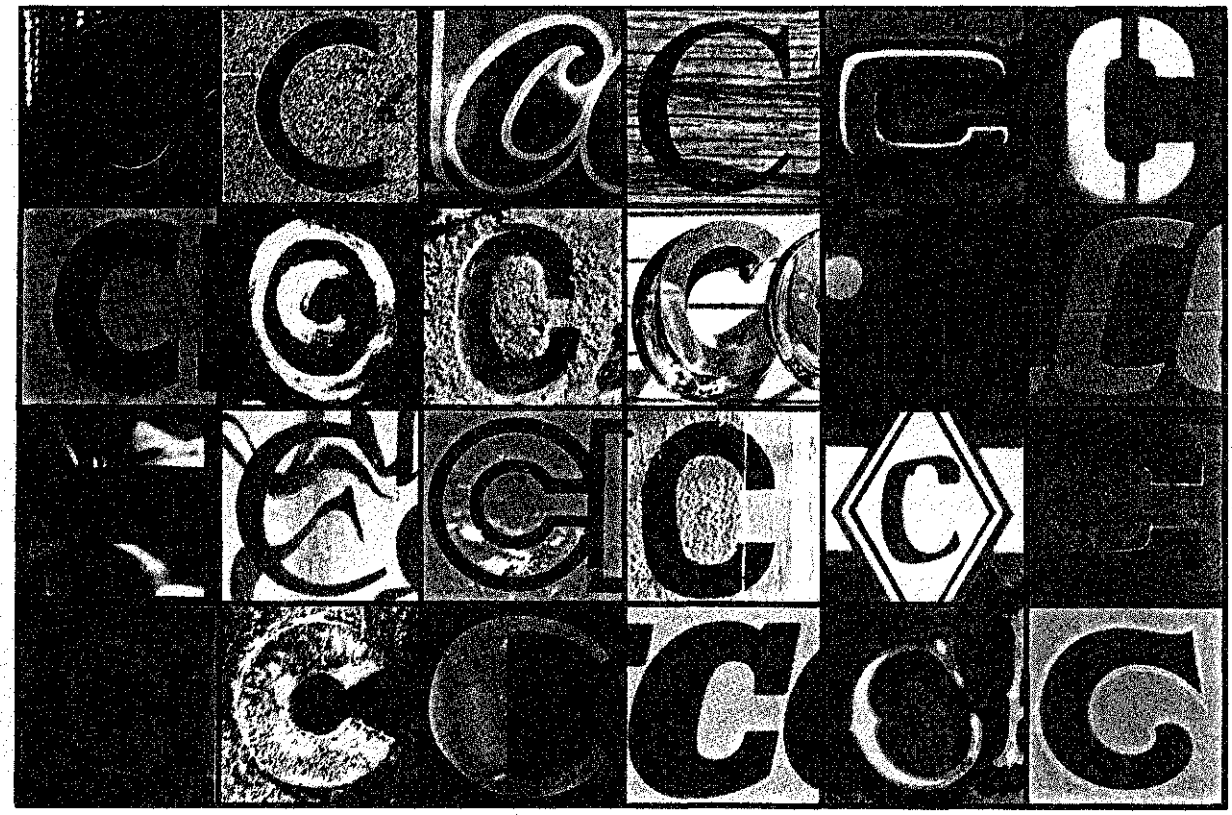
City _____ State _____ Zip _____

Telephone ( )

Return to The Interface Group, Inc., 300 First Avenue, Needham, MA 02194

Number 58 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

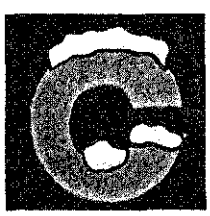


NEW! FROM BLAISE COMPUTING

C Tools Plus™

For The Programmer Whose Alphabet Begins & Ends With "C"

Today's programmers need more than yesterday's tools. Requirements such as removable windows and "sidekickable" pop-up utilities are changing the face of program design. You need to filter interrupts so that other resident programs still work. You need the ability to switch between multiple display pages and monitors. Today's technical demands are almost endless, but C TOOLS PLUS gives you what you need.



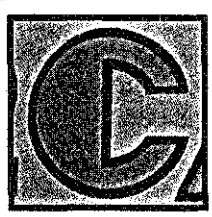
SOLID LIBRARY SUPPORT

Blaise Computing offers you solid library support that can meet all your demands and more. C TOOLS PLUS embodies the full spectrum of general-purpose utility functions that are critical to today's applications.

Here's just part of the PLUS in C TOOLS PLUS:

- **C TOOLS and C TOOLS 2** compatibility—two packages that receive rave reviews for quality, organization, usability and documentation.
- **FULL SOURCE CODE**
- **WINDOWS** that are stackable, removable, that support word wrap and that can accept user input.
- **INTERRUPT SERVICE ROUTINE** support for truly flexible, robust and polite resident applications.
- **MULTIPLE** monitor and display support, including EGA 43-line mode.
- **FAST DIRECT VIDEO ACCESS** for efficiency that will not constrain good program design.
- **DOCUMENTATION, TECHNICAL SUPPORT** and attention to detail that have distinguished Blaise Computing products over the years.

C TOOLS PLUS supports the Microsoft (and IBM) 3.00 and Lattice 3.00 C compilers and is just $175.00.



Also Available Are:

C VIEW MANAGER — A kit for building data entry screens and menus. Begin by designing on-screen what the operator will see; call upon our library functions from your program to display the screens and retrieve the data. Just $275, including all library source code.

C ASYNCH MANAGER — provides the crucial core of hardware interrupt support needed to build applications that communicate. It also includes the "XMODEM" file-transfer protocol and support for Hayes-compatible modems. All source code is included for $175.

C TOOLS & C TOOLS 2 — an indispensable combination still available at a low price of $175, including all source code. See review in PC Tech Journal, *6/85.*

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

ORDER TOLL-FREE 800-227-8087!

YES, send me the PLUS I need! Enclosed is $_____ for C TOOLS PLUS. (CA residents add 6½% Sales Tax. All domestic orders add $10.00 for Federal Express shipping.)

Name: _____ Phone: (____) _____

Shipping Address: _____

City: _____ State: _____ Zip: _____

VISA or MC #: _____ Exp. Date: _____

Number 2 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers

PROGRAMMER'S Journal

Volume 5.1
January/February 1987

Contents

4 **Note from the Editor**

6 **Letters**

8 **PJ Forum:** DTACK GROUNDED, The Junk Mail Flyer *by Hal W. Hardenbergh*
Standards—Yaay! Booo!; graphics hardware; benchmarks; Intel a la Mode

14 **Redirections:** 286 DOS Cometh *by Frank D. Greco*
The protected mode slouches towards your CPU

20 **Consulting:** Marketing and Rules to Live By *by Paul Barkley*
The unavoidable realities

24 **DOS:** Redirection, or They Went That-a-way *by Frank D. Greco*
File redirection in MS DOS, with examples in Assembler and C

36 **On Graphics:** Inside the EGA *by Michael Abrash*
Programming a difficult chip set

48 **Pascal:** How Many Holes Does It Take to Fill an IBM PC? *by Tom Swan*
Efficient string allocation

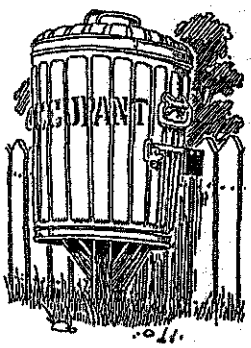
52 **C:** The Proposed ANSI C Language Standard, Part 5 *by Rex Jaeschke*

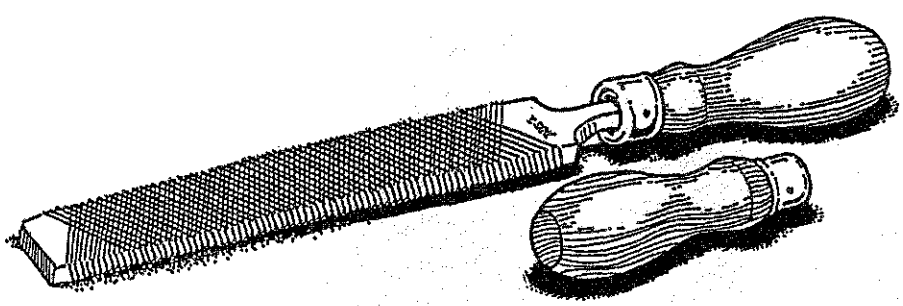
56 **BASIC:** More Tips on Formatting BASIC Data *by Bernard H. Robinson, Jr.*

61 **Misc.**

63 **PJ Billboard**

64 **Housekeeping**





PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers
PROGRAMMER'S Journal

Editor
Robert Keller

Managing Editor
Erin O'Connor

Technical Editor
M. Steven Baker

Advertising & Production
Ralph Bentley

Advertising & Promotion
Robert Tappan

Illustration
Jim Carpenter

Circulation
Gary Hoyer

Publisher
Elizabeth Oakley

Founder
Greg Estes

Contributing Editors
Frank D. Greco, Tom Swan

Contributing Writers
Michael Abrash
Paul Barkley
Frank D. Greco
Hal W. Hardenbergh
Rex Jaeschke
Bernard H. Robinson, Jr.
Tom Swan

Business and Editorial Offices:
Programmer's Journal
29 West 29th Ave., Suite 5
Eugene, OR 97405
(503) 484-2162

Subscription Inquiries:
Programmer's Journal
P.O. Box 3000
Dept. EE
Denville, NJ 07834
(503) 484-2162

Subscriptions:
1 year (6 issues) $24 U.S. ($30 in Canada and $40 elsewhere) 2 years $44 U.S. ($55 in Canada and $75 elsewhere) 3 years $64 U.S. ($80 in Canada and $110 elsewhere)

Articles and letters in Programmer's Journal represent the views of their authors and not necessarily those of the publisher. Letters to the editor become the property of Programmer's Journal and may be edited and published.
Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley or Robin Tappan, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.
IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.
Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Application to mail at second-class rates is pending at Eugene, OR and at additional mailing offices.
POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 3000, Dept EE., Denville, NJ 07834.
©1986 OAKLEY PUBLISHING COMPANY

NOTE FROM THE EDITOR

The New DOS

Several sources have predicted that the developer's version of Advanced DOS, (or DOS 5.0, 286 DOS, or whatever) will appear *real soon now*.

Frank Greco's Redirections column is about Microsoft's operating systems for the coming 286 and 386 protected modes. Also see Hal Hardenbergh's thoughts on the subject (*Intel a la Mode*) in PJ Forum.

Oakley, Bentley Announce Products

In a surprising development, The Oakley Publishing Company unveiled a new peripheral at a November 7th news conference.

"Our R&D department kind of sprung it on us by surprise," said Jim Oakley, cradling the new unit. "But hey, maybe we weren't as clear about corporate plans as we thought."

The device sports an unmodified one-pin male connector. The company indicated that it is robust, but not well-behaved.

Elizabeth Oakley wanted *PJ* readers to know that she wore a Programmer's Journal T-shirt when she carried out the last exciting stages of assembly herself in the wee hours of the morning.

"Now comes the important part," said Elizabeth, "Programming."

And Next, Jacob!

November 13th. In another surprise move, *PJ* Production Manager and artist Ralph Bentley and his wife have unveiled a graphics coprocessor, code-named "Jacob."

The little unit is a hefty eight and one-half pounds. Its optical digitizers can reportedly do graphics computations in less than a second that would take a Cray over 100 years of computing time. Look further for Jacob's first output.

Michael Abrash on Graphics

Michael Abrash returns in this issue with a clear tutorial on programming the EGA. Michael intends to do a series of articles loosely centered around graphics.

The PC display may be the focus of some interesting developments this coming year.

According to Michael, Chips and Technologies shipped a half-million EGA chip sets in their first year, and the total number of EGA boards out there should top 1 million soon.

At Comdex, Intel and Sun showed Sun's beautiful windowed interface (the Network Extensible Window System) ported to and running on a 386 using Intel's new 82786 graphic coprocessor chip. Ray Duncan said in a phone call that the demo looked as impressively fast as the original Sun machine's display.

The 82786 implements bit blits and helps relieve the CPU of the work of maintaining the display bit-map.

Ray predicts that some boards based on the new coprocessor could reach market by March and that if they are in the neighborhood of the EGA in price, they could be popular.

The 82786 doesn't increase screen resolution beyond 640 by 480 as do TI's and Hitachi's recently introduced graphic coprocessors. However, Michael Abrash points out that current mass-market displays really go to only 640 by 480 pixels.

The TI 34010 chip may be competition for Intel. Microsoft reportedly prefers the TI chip because they feel it handles windows more effectively.

Errata

We've gott'em in this issue. A correction to last issue's Turbo inline article is the biggest.

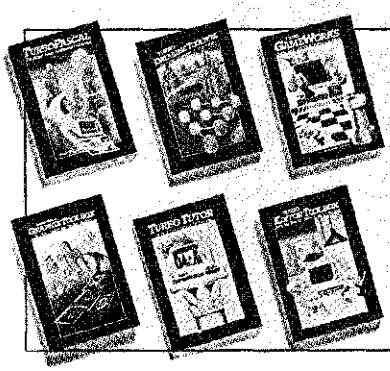
Jacob's first output. See above.

—Robert □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Why more than half a million people are using Turbo Pascal

Because Turbo Pascal is faster than any other Pascal compiler—it's become the industry standard. And because Turbo Pascal is backed by a complete range of "tool boxes" that give you most of the programming tools you'll ever need.



- Turbo Pascal® 3.0 with 8087 and BCD
- Turbo Database Toolbox™
- Turbo Graphix Toolbox®
- Turbo Tutor® 2.0
- Turbo Editor Toolbox™
- Turbo GameWorks®

The Turbo Pascal family is never static, but is continuously expanding, with new products like Turbo Editor Toolbox and Turbo Gameworks.

> "Language deal of the century. . . Turbo Pascal
> **Jeff Duntemann, PC Magazine**
>
> Turbo Pascal has got to be the best value in languages on the market today
> **Jerry Pournelle, BYTE Magazine**
>
> This compiler, produced by Borland International, is one of the best programming tools presently available for the PC
> **Michael Covington, PC Tech Journal**

The secret of software success is not merely low price, but top quality allied with complete documentation, like our 400-page reference manual.

All of which are some of the reasons why Turbo Pascal is clearly the leader, and the recipient of awards like *PC Week's* "Product of the Year" and *PC Magazine's* "Award for Technical Excellence."

NEW! TNT: A DYNAMITE DEAL!
Turbo 'N Tutor
Turbo Pascal *and* Turbo Tutor for only $125.00. You save nearly $15.00, and have everything you need to get up and go with Turbo Pascal!

A whole family of tools

Success breeds success, so the Turbo Pascal family has flourished. Your choices now include:

☐ Turbo Pascal 3.0 combines the fastest Pascal compiler with an integrated development environment, *and* 8087 math co-processor support *and* Binary Coded Decimals.

☐ Turbo Database Toolbox a perfect complement to Turbo Pascal. It includes a complete library of Pascal procedures that allows you to search and sort data and build powerful database applications.

☐ Turbo Graphix Toolbox includes a library of graphics routines for Turbo Pascal programs. Lets even beginning programmers create high-resolution graphics with an IBM,® Hercules,™ or compatible graphics adapter. Does complex business graphics, easy windowing, and stores screen images to memory.

☐ Turbo Tutor 2.0 teaches you *step-by-step* how to use Turbo Pascal, with commented source code for all program examples on diskette.

An offer you can't refuse

You can save $119.75 when you choose the Turbo Jumbo Pack. Six different Turbo Pascal products for only $299.95.

You get Turbo Pascal 3.0 *and* Turbo Editor Toolbox *and* Turbo Tutor 2.0 *and* Turbo Graphix Toolbox *and* Turbo GameWorks *and* Turbo Database Toolbox!

So act now—rush to your dealer or call us today!

NEW! *Amazing value! Turbo Editor Toolbox includes MicroStar, a full-blown editor that also does windows!* You get ready-to-compile source code and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. Also includes MicroStar, a complete editor with full windowing capabilities. (You could pay $100.00 or more for a program like MicroStar, but you get it free as part of our Turbo Editor Toolbox.) You can also use Turbo Editor (which of course integrates with Turbo Lightning™) to build your own word processor!

NEW! *Turbo GameWorks gives you the games you can play, write, rewrite, bend and amend!* Turbo GameWorks reveals the secrets and the strategies of game design. You're given source code, a 200-page manual, and the insight needed to write and customize your own irresistible games.

Also includes ready-to-play Chess, Bridge, and Go-Moku—an ancient Japanese game that can divert you from reality for hours on end.

YES! I want the best!

To order by phone, or for a dealer nearest you,
call (800) 255-8008
in CA call (800) 742-1133

Copies	Product	Price	Totals
___	Turbo Pascal 3.0 w/8087 & BCD**	$ 99.95	$ ____
___	Turbo Pascal for CP/M-80	69.95	$ ____
___	Turbo Database Toolbox	69.95	$ ____
___	Turbo Graphix Toolbox*	69.95	$ ____
___	Turbo Tutor 2.0	39.95	$ ____
___	Turbo Editor Toolbox*	69.95	$ ____
___	Turbo GameWorks*	69.95	$ ____
___	Turbo Pascal & Turbo Tutor	125.00	$ ____
___	Turbo Jumbo Pack*	299.95	$ ____
	Outside USA add $10 per copy CA and MA res. add sales tax		$ ____
	Amount enclosed		$ ____

Prices include shipping to all US cities.

Carefully describe your computer system:

Mine is: ___ *8-bit* ___ *16-bit*

I use: ___ *PC-DOS* ___ *MS-DOS* ___ *CP/M-80* ___ *CP/M-86*

My computer's name and model is: _____

The disk size I use is: ☐ *3½"* ☐ *5¼"* ☐ *8"*

Payment: *VISA* *MC* *Money Order* *Check*

Credit card expiration date ___/___

Card # _____

Name: _____

Shipping Address: _____

City: _____

State: _____ *Zip:* _____

Telephone: _____

CODs and purchase orders WILL NOT be accepted by Borland. Outside USA make payment by bank draft, payable in US dollars drawn on a US bank.

NOT COPY PROTECTED
60-DAY MONEY-BACK GUARANTEE

If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department and we will gladly arrange a refund.

All prices are suggested list prices and are subject to change without notice.

Minimum System Requirements:
Turbo GameWorks, Turbo Graphix Toolbox, Turbo Tutor, & Turbo Editor Toolbox—192K. All other products, 128K.
*IBM PC, PCjr, AT, XT, and true compatibles.
**16-bit only.

TP 36

BORLAND INTERNATIONAL
Vive la différence

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX 172373

BORLAND
TURBO PASCAL
©Copyright 1985. Licensed Material. Program property of BORLAND International, Inc. 4585 Scotts Valley Drive, Scotts Valley, CA 95066. Unauthorized use, duplication or distribution is strictly prohibited by Federal Law.

Borland products include Turbo Prolog, Turbo Pascal, Turbo Pascal for the Mac, Turbo Tutor, Turbo Editor Toolbox, Turbo Database Toolbox, Turbo Graphix Toolbox, Turbo GameWorks, Turbo Lightning, Lightning Word Wizard, Reflex: The Analyst, Reflex for the Mac, Reflex Workshop, SideKick, SideKick for the Mac, Traveling SideKick, and SuperKey—all of which are trademarks or registered trademarks of Borland International, Inc. or Borland/Analytica, Inc. Traveling SideKick is not in any way associated with Traveling Software, Inc. of Seattle, Washington.
CP/M-80 is a registered trademark of Digital Research, Inc. MS-DOS is a registered trademark of Microsoft Corp. IBM is a registered trademark of International Business Machines Corp. Hercules is a trademark of Hercules Computer Technology.
Copyright 1986 Borland International BI-10826W

Number 3 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

New Version 2.0

Complete C Programs in Half the Time, with *Instant-C*™

You can create programs much faster with Instant-C than with conventional programming tools. How? Because Instant-C is a high-performance interpreter, there are **no compile or link delays.** Change your program, then test it immediately. No matter how large your program, the turnaround time is just seconds.

"Instant-C means instant gratification."—*PC Magazine*, **Editor's Choice** for best C interpreter.

Powerful **source-level debugging** saves your time. Conditional breakpoints, single-stepping by statement, source code backtraces, data monitoring, and many other debugging features make it easy to wipe out bugs quickly. Direct execution of any statement or function makes testing a breeze.

"The resulting debugging and testing capabilities are fantastic and the detailed trace/debug/display commands make it easy."—*The C Journal*

Instant-C checks pointer references for reasonableness, and checks that array indexes are within declared bounds. This **run-time checking** stops your program as soon as errors occur, for easiest debugging.

Not only does Instant-C help you quickly change, test, check and debug your code, but it runs your program **fast enough for real-time** applications.

"It is much faster than any of the other products mentioned and was the only one able to complete the standard SIEVE in a reasonable time. Clearly, this high speed allows much more complex problems to be attacked with Instant-C than with any of the other products discussed."—*Computer Language*

Immediate feedback and precise diagnostics make Instant-C great for learning C. Full K&R and the ability to **link compiled object code and libraries** (Lattice and Microsoft) makes Instant-C compatible with your existing programs.

Instant-C makes all parts of the programming task as fast as possible.

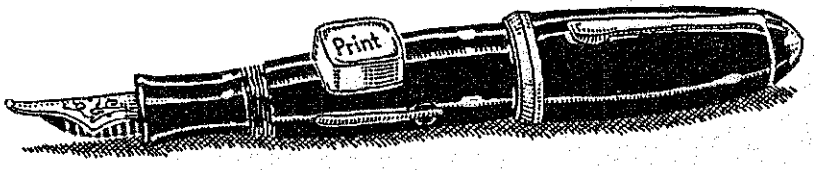
"Clearly, Instant-C is the performance champion."—*PC Tech Journal*

Version 2 works with MS-DOS and PC-DOS, and has a full 31 day **money back guarantee.** Instant-C is only $495. Order today! Call or write for full information.

Rational Systems, Inc.
P.O. Box 480
Natick, MA 01760
(617) 653-6194

Number 20 on the Reader Service Card

LETTERS



From Mnematics, *PJ*'s Bulletin Board

FNE says (Volume 4 Number 2) that the first 16-bit microprocessor was by National Semiconductor in 1974. Wrong, wrong. The first was General Instrument's CP-1600, a PDP-11 on a chip. This P-MOS (requiring positive and negative supplies) chip put in a pretty impressive performance at 4 MHz in 1972. Although never actively marketed, the chip was used in Mattel's Intellivision video game. The chip is still in production today to the best of my knowlege. The chip was developed at Honeywell, and when Honeywell tried to bury the project, the engineers took their design to General Instrument. GI found a customer for it and built it. Two years later, I designed the second generation development system for the beast.

I'm very interested in Hal H's rantings about the Fairchild Clipper. Hal is a bit off the track in two places, though:

1. The lead time for incorporating new semiconductor devices into products and getting them to market is stretching out. This will affect the timings which are the basis of Hal's vision(s).
2. Motorola and (unfortunately) Intel are not sleeping while Fairchild plays CPU architect.
3. I remember Fairchild's last great CPU, the F8. I wonder if anyone else does.

Gregg Squires

Yo!

I saw an F8 today. We still use Z80 based machines in the department. We now have two Atari 1040 ST's though they are of no use to man nor beast as they are somewhat impossible to interface to the outside world . . .

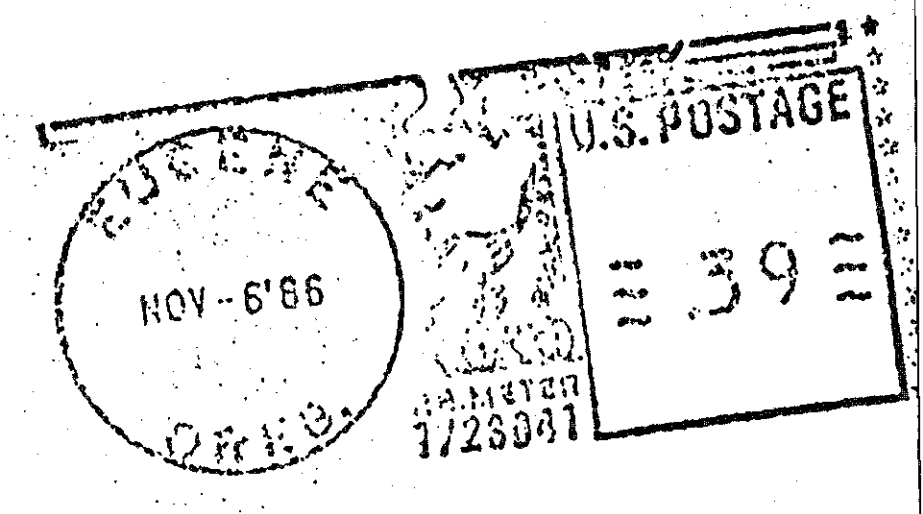
As for programming, does anyone have a decent command parser that can cope with REAL ENGLISH? I was outlining one the other day . . . works on KEYWORDS + ORDER.

Cheers,
Peter Evans, –7899

Hi,

Believe it or not, we still make a high voltage power supply that uses the F8. The guy responsible for the code went nuts hand-coding the thing and wrote an assembler for it!!

New User, –6440 □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Experience Prolog

with

APT™

Active Prolog Tutor

Learn Prolog While Building Applications Interactively

A P T is *designed* to teach you. With the guidance of the text and the support of the Prolog interpreter, you can experiment with new concepts as they are presented.

Use a Special Prolog Interpreter

Prolog concepts are shown in text form and then demonstrated by example using the Prolog interpreter. You can also use the interpreter as a practice environment to try ideas and get immediate feedback. A single keystroke toggles you between the tutorial text and the practice interpreter. You can write a program and execute it without leaving the tutorial environment.

The practice interpreter is designed for education and is integrated with the tutorial package. The interpreter is not appropriate for production applications.

See for Yourself How Prolog Works

Traditional teaching tools have been ineffective in presenting what happens as Prolog executes. You need to know *how* Prolog behaves in order to write useful programs. A P T has 3 tracing mechanisms which will literally show you:

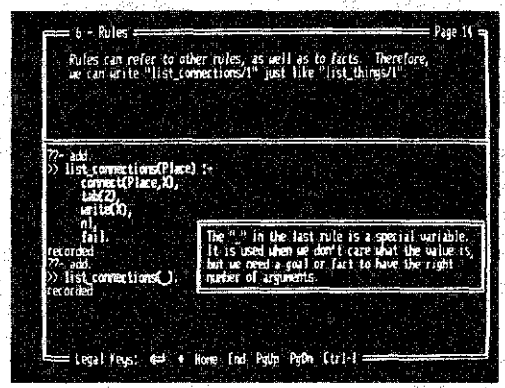
--the execution pattern of Prolog, and clause order importance.
--how values are passed up and down in a recursive Prolog routine.
--how a Prolog query is processed.

Investigate Several Applications

A P T is a self-contained practice environment. It guides you through the development of:

- an adventure game
- an intelligent genealogical database
- an order entry system
- an identification expert system
- a natural language user interface

Each program builds on the one before. By the end of the tutorial, you have actually built working Prolog applications.


6 - Rules
Page 14
Rules can refer to other rules, as well as to facts. Therefore, we can write "list_connections/1" just like "list_things/1".
?- add.
:- list_connections(Place) :-
connect(Place,X),
write(X),
nl,
fail.
recorded
?- add.
:- list_connections(_).
recorded
The "_" in the last rule is a special variable. It is used when we don't care what the value is, but we need a goal or fact to have the right number of arguments.
Legal Keys: ⇐ ⇒ Home End PgUp PgDn Ctrl-I

A common APT screen will include a tutorial description of what is going (top window), interpreter code (bottom window), and a pop-up window with helpful hints. Legal keys are always displayed on the bottom of the screen.

All the Extras You Need

- Manipulate the tutorial with single keystrokes
- Move to prior pages easily
- Instantly access any chapter with the table of contents
- Helpful hints appear in pop-up windows
- Keys you can use are displayed at the bottom of the screen
- Automatically begin where you left off
- All work is automatically saved when you leave the tutorial
- The Prolog snytax you learn is compatible with:
  --The Clocksin & Mellish standard
  --Prolog 86 and Prolog 86 Plus by Solution Systems
  --Arity Standard Prolog and Arity Compiler/Interpreter
  --Also compatible with Turbo Prolog's version of Prolog syntax

"APT is fun to use. It is very highly interactive. At almost anytime, you can terminate the formal learning process and play around. This is THE introduction to Prolog."

—Larry Lidsky
Professor of Nuclear Engineering, MIT

System Requirements

APT requires an IBM PC, XT, AT or compatible hardware and PCDOS 2.0 with a minimum 512K.

MONEY-BACK GUARANTEE
Try Active Prolog Tutor ($65) for 30 days — if not satisfied get a full refund.
TO ORDER CALL (800-821-2492)

Solution Systems™

SOLUTION SYSTEMS, 335- J WASHINGTON ST., NORWELL, MA 02061. 617-659-1571

Number 47 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DTACK GROUNDED, The Junk Mail Flyer

by
Hal W. Hardenbergh

Exeunt Eagles:
After three years and four months of faithful service, our two Eagle IIs have been retired. Nothing wrong with them (neither of them so much as hiccuped the whole time), but they ain't compatible with our HP LaserJet+. This is being typed at home using WordPerfect 4.1. Digital Acoustics bought just the one copy of 4.1 for me, and it is licensed for just one machine. That means I have to carry my IBM PC home with me every night and back to work every morning. If I maintained a separate PC at home and just carried the diskettes back and forth I would be a software pirate. Tsk.

I have had some recent experience with a turbo PC clone. At 8MHz, WordPerfect is not so noticeably slow as it is at 4.77MHz. I can see why folks are buying ATs to run PC software. I think I'll buy myself an AT this fall when the rest of the new-generation stuff comes out. The artistic renditions of the Tandon AT clone interest me, especially that nice-looking front porch on the keyboard to rest the heels of one's hands on.

Another reader suggested XY Write, which he asserts is written in macro assembler and hence is much faster than Word Perfect. Two days after I received that letter, *PC World* showed up with a review of XY Write 3. I sure would like something faster than Word Perfect . . .

Standards Are Wonderful.
Standards Are Awful.
Both of the above are true. The American National Standards Institute (ANSI) came into being because the nuts and bolts used to hold one automobile together were not interchangeable with any other auto's nuts and bolts. Now one 6–32 screw is interchangeable with any other, which is wonderful.

If you buy an Apple IIe for your kid today so that he can run the same software as the older kid next door who got his Apple II+ several years back, you will be happy to know that the disks created on an Apple II+ several years back can be loaded and run on today's IIe. That's wonderful. On the other hand, the disks on today's IIe are ludicrously out of date compared to today's technology. And that is why sales of the IIe are finally tailing off. That's awful.

Another reason that the Apple II's sales are dropping off is that its display, which in fact represented state-of-the-art technology when the Apple II was introduced, is miserably inadequate in 1986, especially for text. But most Apple II programs are locked into the specific hardware configuration used by the original Apple II. Improve the text display and you can take all those programs and trash them. Yet another Apple problem-standard is the Macintosh's display, which was obsolete the day it was introduced.

When IBM introduced its PC in the fall of 1981, it was supposed to use cassette tapes for program and data storage. That phase lasted an entire five seconds before floppy disks became the standard storage devices for the PC. In early 1982, 180K was the most data you could reliably put on the then-standard single-sided 5.25 inch diskette. A year or so later, as sales of the PC were ramping up, a double-sided version of that disk drive appeared that would hold 360K, and that became the standard. So in 1986 the standard diskette on the PC is still 360K, and that will doubtless continue to be true until, as with the IIe, folks stop buying PCs.

In 1986 floppy disks—economical floppy disks—hold four times as much data as the PC's 360K standard and nine times as much as the IIe's 140K standard. Hardly anybody uses these high-density (HD) floppies, not even on the IBM PC AT, which comes with one such disk as standard equipment. What gets used on ATs is either a hard disk or a—you guessed it—360K standard floppy. If I had a floppy with four times more storage (and I do), I believe I would be tempted to use it (and I do). But not the PC folk.

In July 1981, I sat down at my drafting table and with James standing by kibitzing, I defined the architecture of a synchronous 68000 attached processor. Motorola intended the 68000 to be asynchronous; grounding DTACK makes it both faster and also synchronous. Because this board was to fit inside a Commodore 8032 Pet, it turned out to be 6.5 by 15 inches. Because I wanted to hang graphics boards and math accelerator boards onto it, I defined an I/O expansion port. James and I spent at least 35 minutes making these decisions. So Digital Acoustics built the thing, and as you know, it worked.

Today is 1986. I am laying out a board to hold a 68020, a 68881, and one to ten megabytes of RAM. It is 6.5 by 15 inches in size. The expansion connector is identical to, and in the exact location as, that 1981 design. And I will continue to build boards that size with that expansion connector until nobody buys them anymore. It's the Apple II disk and display problem

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

all over again. Standards are wonderful. And awful.

Sometimes a standard becomes so outdated that it has to be changed. This is usually accompanied by much shouting, shrieking, wailing, and rending of garments. Two such changes are in progress at Microsoft. One is the 640K memory limitation imposed by MS DOS. Bill Gates, who desperately needs a no-man, is moving that limit all (?) the way up to 16 megabytes. That means MS DOS will have severe memory limitations again a mere four years hence. The other is the 30 megabyte limitation on hard disks.

There is another MS DOS standard which desperately needs to change but which won't. That is the standard of using INPUT and PRINT statements to fetch and store numeric data from and to disk. You may be interested to know that MS DOS uses INPUT and PRINT in 1986 because RAM was horribly expensive in 1969. Not 1979, 1969. That is a true statement. Sometime when I have lots of space I'll explain it.

All together now, let's hear it for standards: Yaay! Booo!

Jim Seymour Watch:

Jim is now the overwhelming favorite to win the 1986 Jeanyates award. In Aug 5 *PC WEEK* he correctly reported that "Windows is a slug on 8088 PCs, an impossibility on floppy-disk PCs. . ." and yet he also asserts that "I think developers who don't build Windows compatibility into new products and new releases of successful existing products are crazy."

Hmm. Windows is a slug, and anybody who doesn't use it is crazy. Contradictory? Well, it seems that Jim has finally arrived where Carl Helmers was four years ago. Carl was the guy who told Jerry Pournelle, with regard to writing LISA's operating system in slothful Pascal, "the CPU is so fast it doesn't matter!" As you all know, Carl was correct and LISA proved to be a smashing success and dominates the personal computer marketplace today.

Similarly, Jim's reason for asserting that Windows is a slug and anybody who doesn't use it is crazy is that the 80286 runs the EGA display so fast it doesn't matter!

You know, if this attitude begins to permeate the IBM arena, I think I might buy some Apple and (if available) Atari stock. Those are the only personal computer folk not flogging PCs or PC clones. (For the one reader who did not catch on, I was being sarcastic about LISA's "success." When you deliberately bust your hardware's performance, you are on your way out of business.)

Some publications—*InfoWorld*, for instance—have begun to notice that the reason folks buy ATs instead of PCs is performance.

If Jim asserts that all the new 386-based machines which will be announced this year will run Unix, he will win the 1986 Jeanyates Award by acclamation!

A Window to the Future:

In a few years—four or five—graphics hardware will have high enough performance and low enough cost that windowing environments may become practical without slowing things to a crawl. I am referring to dedicated graphics processors that offload the work from the CPU. The EGA board Jim Seymour refers to has no on-board smarts at all.

Meanwhile, some of the other companies that announced Hitachi 63484-based graphics boards a year ago have by now discovered the "feature" which was first documented in a March 1986, 63484 manual and which requires that two inexpensive LSTTL chips be added to a 63484 board before it will work. (That manual doesn't mention those two new chips directly; one is supposed to infer them.) According to our local Hitachi rep, we (me and James) were the first in Orange County to discover and fix that little "feature." Which is amazing because a lot of graphics boards and such are designed and built here in Orange County, CA.

Yesterday afternoon I watched as James exercised the Atari ST using resident, let me repeat that, resident HALGOL v.04. The performance is just as snappy as that of a DTACK/host system, and as with the DTACK/IBM system, one has a real 80 x 25 upper/lower case display. The scroll rate was astonishingly fast for a bit-mapped display (but then we were working with pure text, no windows). Those of you who own STs and are dismayed by the abysmal performance of ST BASIC: Rejoice! The problem is the software, not the hardware!

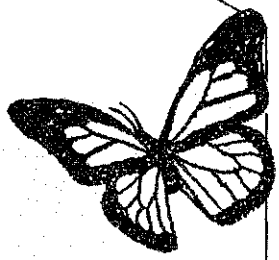
We used a crude, inefficient fix for the zero page problem (us DTACK folk are the only ones in the world who have a correctly-designed zero page in our 68000 hardware). So there

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Chalcedony PROLOG

Not Copy Protected

A REAL Clocksin & Mellish Prolog for BOTH major microcomputing operating systems — with full cross-compatibility. $99.95



Complete with the predicates necessary for POWER AI programming:
op () name () functor () clause () =..("Univ")
...And no constraining data typing.

- Floating point
- Step-by-step tutorial
- Math functions
- Integrated editor

PROLOG/i

Extensible overlay library, 8087 support, large memory model (up to 640K)

PROLOG/m

Complete Macintosh environment with extensive pull-down menus and dialogue boxes.

No Risk Offer: Examine the PROLOG/i or PROLOG/m documentation at our risk for 30 days. If not completely satisfied, return with disk still sealed for refund.

APPLICATIONS — *Complete with SOURCE CODE*

NFL X-pert $49.95
A true interactive expert system written by a professional knowledge engineer. A valuable learning tool for any Prolog programmer interested in using Prolog to develop expert systems.

TOOLBOX $29.95
More than 50 subroutines that speed and compress list handling, searches, sorts, and reversal algorithms. An inside look at the tricks of the professional Prolog programmer.

TOYBOX $29.95
Written by an academician to help his students understand Prolog, this collection of puzzles and mind-teasers will illustrate how the Prolog programmer creates programs that find the best solution to the problem. Turn your computer into a super reasoning machine!

CHALCEDONY SOFTWARE, INC.
5580 LA JOLLA BLVD.
SUITE 126
LA JOLLA, CA
92037
(619) 483-8513

System Requirements:
Minimum 256K RAM (320K recommended) PC DOS/MS-DOS ANSI Standard Support
Minimum 512K Macintosh Macintosh-plus and HFS Compatible

SAVE 10% when you buy either PROLOG/i or PROLOG/m and all 3 applications.

PHONE ORDERS: 1-800-621-0852 EXT 468

☐ PAYMENT ENCLOSED $ _____
CA residents add 6% sales tax
☐ CHARGE MY: ☐ MasterCard ☐ Visa
Card No. _____ Exp. Date _____
Signature _____
Mr./Mrs./Ms. _____
(please print full name)
Address _____
City/State/Zip _____

PROLOG/i	$99.95
PROLOG/m	99.95
Check:	
MS-DOS ☐	Mac ☐
NFL X-pert	49.95
TOOLBOX	29.95
TOYBOX	29.95
Complete Pack	188.82

SHIPPING:
$ 5.00 U.S.
7.50 Canada
10.00 Carribean, Hawaii Air
20.00 Overseas Air

Number 64 on the Reader Service Card

is about an extra 15% overhead in currently-resident HALGOL. Nevertheless, the BYTE Sieve benchmark ran in 33.3 seconds using resident HALGOL v.04 vs. 960 seconds using Atari ST BASIC. That's 29 times faster, which is about one and a half orders of magnitude. A Concorde supersonic jetliner cannot quite fly two orders of magnitude faster than a normally athletic 13-year-old girl can run . . .

Good Hardware Dept:
About a year ago we took a chance and bought a newly-introduced high-speed dot-matrix printer, the OT-700, made by Output Technology Corp in Oregon. We now have six or seven FX-80s, one OT-700, and one HP LaserJet+. Guess which gets used the most? Wrong! The OT-700 gets used the most! We use that printer at a dedicated printer station, and we run the hell out of it. With James porting HALGOL v.04 to the ST and me developing HBASIC, that is a very very busy printer. It just sits there, routinely kicking out 200 lines a minute by my timing. I still want a 1200 lpm printer for Xmas, but until Santa comes, the OT-700 does the job for us. If you need fast, inexpensive z-fold printout, the OT-700 will do the job for you even if you use 13-inch forms. No, OTC didn't pay me for this ad; I just think good products should be acknowledged.

If you do not need z-fold and you can get by with about 150 pages a day maximum, the LaserJet+ is pretty neat, to understate the matter. But it is not small, and it is not cheap.

Benchmark Stuff:
Somebody once said, "There are lies, damned lies, and benchmarks." Most companies do not like benchmarks for the simple reason that, in a competition with 100 other companies, the odds are 100–1 against their winning. According to a lengthy letter that appeared in *BYTE* magazine several months back, Intel cheated a little (ahem!) bit on a recent benchmark comparison against Motorola. I have some very specific information about a particular major microprocessor manufacturer's benchmark methods and policies which would curl your hair. Unfortunately, I can't tell you about it because to reveal the information would also reveal the source. I was given that information off the record.

And then there is the minicomputer company, one of the prime mini makers in fact, which has pulled off a cute trick. A benchmark which is commonly used on superminis is the Whetstone. This benchmark mostly checks floating point performance, but it also checks looping ability. There is some code which could be done as in-line code but which the Whetstone code deliberately puts in a loop. This certain mini producer has modified its compiler so that the compiler can recognize the Whetstone code. When it recognizes that code, it ignores the loop portion and compiles straight-line code. . .

I have written about all of this stuff in the past. The reason I bring it up now is that it is hitting close to home.

HALGOL was a language in which general expressions were allowed only in assignment (LET) expressions. Most statements, aside from LET, permitted only a simple varn or constant of a given data type as an argument. BASIC, on the other hand, mostly allows general expressions of a given data type as arguments. Programming HALGOL was therefore different from programming BASIC in that any general expression had to be evaluated in an assignment statement before being used elsewhere.

Lots of you readers suggested that such a language could never be a commercial success. I agree, which is why I have

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

been busy of late implementing general expressions. One of the things about general expressions is that an array element index is determined by a general expression which can contain another array element, which . . . in other words, we can have nested arrays until the end of time or the end of the input buffer, whichever comes first.

Naturally, the way one handles this situation is to write reentrant code for the subroutine which returns the address of an array element. I did. It works. So I wrote the SIEVE benchmark in this newly-general language, now known as HBASIC. It ran in 24.5 seconds. I re-loaded HALGOL, and the same algorithm still ran in 19.5 seconds on the same 12.5MHz DTACK board. Hmmm.

So I wrote a simple benchmark in both HALGOL and HBASIC:

HALGOL:

```
DIM A%[32000]:ONE%=1:FOR I%=0 TO 31999:
MVA ONE% TO A%[I%]:NEXT I%
```

HBASIC:

```
DIM A%[32000]:FOR I%=0 TO 31999:A%[I%]=1:NEXT I%
```

HALGOL ran that in 1.388 seconds. HBASIC required 1.884 seconds. That works out to an extra 194 clock times per array element assignment. So I pulled out my newly-wonderful code, the part which is wonderfully reentrant, and compared it with the old code. I very quickly discovered where 172 of those 194 extra clocks came from. Most of them were associated with the necessity to store intermediate results on the stack, which is how one writes reentrant code. Some were associated with the fact that generalized HBASIC compiles into two additional action addresses, and each action address word has 26 clocks of runtime overhead just in the threaded-code kernel. (To keep this in perspective: 26 clocks at 15MHz, which is the speed of the DTACK board we are now shipping, is 1.73 microseconds. Those 194 extra clocks per array element assignment are 12.9 extra microseconds.)

In other words, one has to pay a penalty for the luxury of having perfectly general expressions on tap at all times.

But an array which only has one dimension does not have to keep intermediate results on the stack because, in computing the value of only a single index, it does not have to retain the value of previous indices. So a reference to a singly-dimensioned array could be compiled to a separate, non-reentrant, routine to evaluate the address of the array element. And if the index of that one-dimensioned array is a simple integer varn or constant, then it could be compiled to a special form of code which is exactly like HALGOL and which does not use those two extra action addresses that would otherwise be required. (I think this is called peephole optimization.)

And then SIEVE would run in HBASIC just as fast as in HALGOL because, for the simple array references the SIEVE benchmark uses, HBASIC would be HALGOL, not HBASIC!

While perfectly general expressions are indeed working now, they have been integrated only into the HTAB, MID$(and VTAB statements (aside from LET). Now the FOR, IF, PRINT, etc., statements must be changed to incorporate those general expressions at compile time, at run time, and at list time. Meanwhile, James is porting HALGOL v.04 to the ST as a resident program—he had already ported it to the DTACK board using the ST as a host, something I forgot to mention in the last junk mailer. So when I finish incorporating general expressions into all the statements that one expects a BASIC to have, James will already know how to port HBASIC to the ST as a resident program. And that means that (tadaa!) we will have a software product to monger!

See how simple it is to write a commercial BASIC for a new machine?

More Benchmark Stuff:

(LATER:) After some peephole optimization and a little fine-tuning, HBASIC ran the Sieve in 19.192 seconds on a 12.5MHz BG. Here's how we got that 19.192 seconds: We checked array elements for their number of indices. If there is only one index, there is no need to use a reentrant form of array element address calculation. If there was only one dimension, we further checked to see if the index was just an integer varn (as is the case in the Sieve benchmark). If so, we switched the compiled code to yet another form of array element address calculation, one which is optimized for an integer varn index and which has two fewer action addresses in the runtime code. That got HBASIC's performance back to that of HALGOL.

That exercise sensitized me to the fact that every little clock cycle counted. So I went clock-cycle hunting. I noticed that almost all of the integer arithmetic routines ended:

```
BVS IOVFL
JMP (A4)
```

I changed that to:

```
BVC HBASIC
BVS IOVFL
```

where HBASIC is the same address as is contained in A4. That saves six clock cycles. I probably should reexamine the floating point routines, which are nearly five years old now, because I've learned a lot about the 68000 instruction set in those five years. But those routines are thoroughly debugged, and I have lots of other stuff to program. (Come to think of it, I did reprogram the divide and multiply a couple of years back, and yes, the new routines ran faster.)

Most people who program in BASIC seem to believe that the first dialect they were exposed to is the natural order of the Universe, with all other dialects being the work of the devil. I am no exception. My first BASIC was the one which ran on my Wang 2200. One of its neat features was the ability to INIT(ialize) all of the elements of an array to a particular value. So I looked at that Sieve benchmark and noticed where the ability to initialize an integer array to the value 1 might speed things up a tad. I checked the BASICA manual, the ST BASIC manual, and a couple of other BASIC manuals, but none of them had the INIT statement. So I consulted with James and we invented our own INIT syntax:

```
INIT A[]=(fp expression)
INIT A%[]=(int expression)
INIT A$[]=(string expression)
```

So I replaced the following line of the Sieve:

```
FOR I%=0 TO 8190:FLAGS%[I%]=1:NEXT I%
```

with (hello there compiler which straightens loops!):

```
INIT FLAGS%[]=1
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

And HBASIC now runs the Sieve in 16.15 seconds (12.5MHz). That corresponds to 25.2 seconds on the ST. By comparison, Softworks Limited's SL BASIC compiler takes 40 seconds, and ST BASIC takes 960 seconds! Understand, we were whipping the SL compiler pretty badly even without the INIT statement, but I have newly rediscovered the truth that every little bit helps, and every little clock cycle counts. (Hmm . . . I wonder if a few clocks can be trimmed from the integer FOR-NEXT runtime code?)

Why is SL's compiled BASIC so much slower than interactive HBASIC? I keep telling you compatibility fans that when you are compatible with a donkey cart, what you have is a donkey cart! SL's compiled BASIC is ten million per cent compatible with Alpha Micro BASIC. While Alpha Micro BASIC is not a donkey cart, neither does it do true integer arithmetic. BASICA doesn't either. HBASIC does.

There are lies, there are damned lies, and there are benchmarks . . .

James gave me a printout of SL's canned SIEVE program. I thought it was Pascal, the way all the lines were indented and with not a line number in sight. All of you folks out there who want BASIC programs that look just like Pascal really should buy SL's BASIC, provided compilers with up put can you. I can't.

So why did Digital Acoustics buy SL's compiler? To get a look at the competition in part, but mostly to see how they bound the manual. I may use their spiral-binding technique on the first release of HBASIC, although HBASIC Release 2 will definitely have a Borland-type manual. SL's manual looks nice. Remember that SL's BASIC has been around for a while; it's just new on the ST.

David Cortesi Watch:

David Cortesi, who no longer writes a column for Dr. Dobb's, once reviewed Turbo Pascal. It would be more accurate to say that he savaged it! It seems that Turbo Pascal (installed base 300,000+) does not comply with ISO Pascal (installed base 17). Since I am currently defining a language, I was especially bemused by David's bitter complaint that Turbo Pascal followed Niklaus Wirth instead of ISO on one particular point!

Softworks Limited's chief programmer was discussing this with me. He pointed out that software reviewers do not review software! What they do is review what they wanted. What David Cortesi wanted was ISO Pascal, and so his review of Borland's product was in fact a review of whether or not it followed the ISO standard. (ISO = International Standards Organization, which is not legally recognized in this country because we have ANSI = American National Standards Institute.) Actually, ISO Pascal s . . . , uh, ISO Pascal is not so hot. It neither follows the rules established by Pascal's designer nor fixes the "thalidomide baby" problems that original Pascal had. As a useful language, Turbo Pascal is much better than either original Pascal or its ISO version. That might have something to do with why Borland has sold over 300,000 copies!

I just read a two-page review of Philon BASIC, which is another compiled BASIC for the ST. The reviewer complained that Philon used CLS rather than CLEARW2 to clear the screen and home the cursor. CLS is BASICA dialect (HOME is Fruitysoft][dialect), while CLEARW2 is ST BASIC dialect.

Dick Barnes, who is the editor of the Amigan newsletter, is a fan of lightly structured BASIC. Not massively structured like Pascal, mind you. So Dick will probably not ever like HBASIC because HBASIC will never be structured. (Dick does like HBASIC's speed, which he noted several times in his late SuperPet newsletter.)

The thing is, different folk want different things (remember Santa Ana Squares?). What a language designer has to do is decide what he wants to do and establish a set of language specifications. A language reviewer most certainly should note whether the specifications are reasonable as a part of the review but should not go the extreme of blasting the specifications for not conforming to the reviewer's desires, as David Cortesi did with Turbo Pascal. Another thing which should be reviewed is how well the final product complies with the specification.

Intel a la Mode(s):

Watching the IBM PC MS DOS folks these days is more fun than a barrel of monkeys. There is fierce thrashing about as attempts are made to overcome the 64K segment limitation and the 640K MS DOS limitation and the 1 megabyte 8088 limitation at a time when megabytes, plural, of RAM are downright cheap.

Intel thoughtfully built an 8088 emulation mode into the 80286. To emulate an 8088, a megabyte is the most you can address. And to run MS DOS, you are still limited to 640K. Because so many programs are available for the MS DOS environment, the 8088 emulation mode is the one used by all the 80286-based machines all of the time. (There may be 17 folks nationwide who use the other mode of the 80286. Then again, there may not be. It turns out that my prediction that 17 folks were going to buy UNIX PCs for their own personal use was over-optimistic!)

You will be pleased to know that the 80386 also has two modes. One is an 8088 emulation mode which has the same addressing limitation as the 8088. The other is the real 386 mode, the one with 32-bit addressing, which is the one which will not be used in the personal computing world. Where the real 386 mode will be used is in low-end minicomputers and in engineering workstations. Software wears humongous price tags in either of those environments. But I write about the personal computer world, so the following is restricted in scope to us individuals running our own computers:

The 386 has two modes, only one of which will be used. The 286 has two modes, only one of which is used. Bill Gates is busy pushing a version of MS DOS which uses the real 286 mode onto a reluctant bunch of software vendors. Them reluctant folks has suddenly switched into extremely reluctant mode! You see, they just discovered that although both the 286 and the 386 is compatible with the 8088, ain't neither the 286 or the 386 compatible with the other! And that means that an investment in a new, complicated operating system which uses the real 286 mode will be a total loss as soon as the 386 machines arrive! Talk about narrow marketing windows! He he he!

An OS Observation:

Has anybody else noticed that none of the new window/icon/mouse operating systems is making inroads into the time-is-money personal computing environment? And there are lots of those systems out there: Desqueview, TopView, Windows, the new text version of Windows, GEM . . . and none of them is making it. When you see folks mousing around, they are usually holding a cocktail (Perrier and water) in their left hand. Not time-is-money!

There is one emerging environment that does use mice and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

bit-maps in a time-is-money environment. That is the new desktop publishing environment. Bit-maps, mice, and icons really are the best way to mix text and graphics. Meanwhile, the Macintosh is evolving more and more into a real (if overpriced) computer. If this trend continues, I will probably buy the next model. In January?

Don't laugh, I'm not moving in Mac's direction. Mac has been moving in my direction. That first northward-only wimp-Mac was and is a piece of (deleted). Macintosh sales, which had been running only 13,000 units per month including the last XMAS shopping season, have jumped to 20,000 per month as desktop publishing gathers momentum (and as the Mac+ gets closer to being a real computer). But Mac is being outsold about 8–1 or 10–1 by MS DOS machines. It seems that most folk prefer a real keyboard and a real 80 column text display sized for adult eyes. Oh, yes: Mice need not apply. Unless you are doing desktop publishing.

The Next-generation 680X0:
It takes roughly three years to move a microprocessor from announcement to mass production. A year from announcement to first silicon, a year from first silicon to reasonably bug-free pilot production runs, and then another year to true mass production. Obviously, this is approximate.

You don't bother with a new microprocessor design unless it is about twice as fast as the older part. Which means that performance doubles every two years. Part of that performance gain is from a faster clock and part from a more complete and efficient architecture. Take the 68000 and the 68020, for instance: They are two generations apart, so the 68020 should be four times as fast as the 68000, and, for nominal clock rates, it is. A 16MHz 68020 is four times faster than the 8MHz 68000. So half of that performance gain came from a faster clock.

Somebody—perhaps several somebodies—is right now designing the architecture of 1989's 68040. The 040 will use 1.1 micron design rules, have a 33MHz standard clock rate, and will require four times the memory bus-bandwidth of today's 68020. I'm glad somebody else is figuring out how to most efficiently accomplish that!

1991's 68050 will need a 128-bit memory width interleaved into the 050's physical 64-bit data port. When combined with static CAS mode, that will provide enough bus-bandwidth to accommodate the 050's 50MHz clock rate, made possible by 0.8 micron design rules. Hey! What are you guys doing? What are those straps for? No! Hey, I'm fine! Really! Put me down, you guys. . .

Bragging Dept:
The recently concluded U.S.-Japan trading negotiation, according to the *WSJ*, "includes an informal understanding that Japan will take steps to allow foreign microchip makers to increase their share of the Japanese market to 'more than 20%' by 1991. . ."

Informal understanding? Gosh, you mean they didn't put it in writing? Who could ever have predicted that?

The *WSJ* also reported: ". . . the Japanese government agreed to police its own producers to guard against predatory pricing either in the U.S. or in third-country markets." I suggest you read that last sentence again. Nobody's going to guard against predatory pricing in Japan!

Later: Some folks have not read the *WSJ* carefully. Jack Robertson in his regular *Electronic News* column seems to think that the Japanese promised the U.S. chip makers a 20% share of the Japanese market by 1991. Wrong! They promised foreign chip-makers a 20+% share: 15% to Korea, 2% to Germany, and 1% each to France, Italy, the U.K., and the U.S. counts. . .

If any of you readers out there thinks this recently concluded agreement is going to stand up for as long as one year, I want you to mortgage the farm, sell the family jewelry, convert everything into cash. Then come see me; I have some wonderful investment opportunities just for you. □

Here's why you should choose Periscope as your debugger...

You'll get your programs running fast. "It works great! A problem we had for three weeks was solved in three hours," writes Wade Clark of MPPi, Ltd.

You'll make your programs solid. David Nanian says, "I can't live without it!! BRIEF, a text editor my company wrote, would not be as stable as it is today without Periscope."

You'll protect your investment. We won't forget you after the sale. You'll get regular software updates, including a FREE first update and notice of later updates. You'll get technical help from Periscope's author. And you'll be able to upgrade to more powerful models of Periscope if you need to. One Periscope user writes, ". . . your support has won over even the heart of this hardened programmer!"

You deserve the best. Thousands of programmers rely on the only debugger that PC Tech Journal has ever selected as **Product of the Month** (1/86). You owe it to yourself to find out why, first hand.

You can try it at no risk. You get an unconditional 30-Day, Money-Back Guarantee, so you can't lose.

Start saving time and money now — order toll-free, 800/722-7006. Use MasterCard, Visa, COD, or a qualified company purchase order. As one user puts it, Periscope is "one of the rare products, worth every penny!"

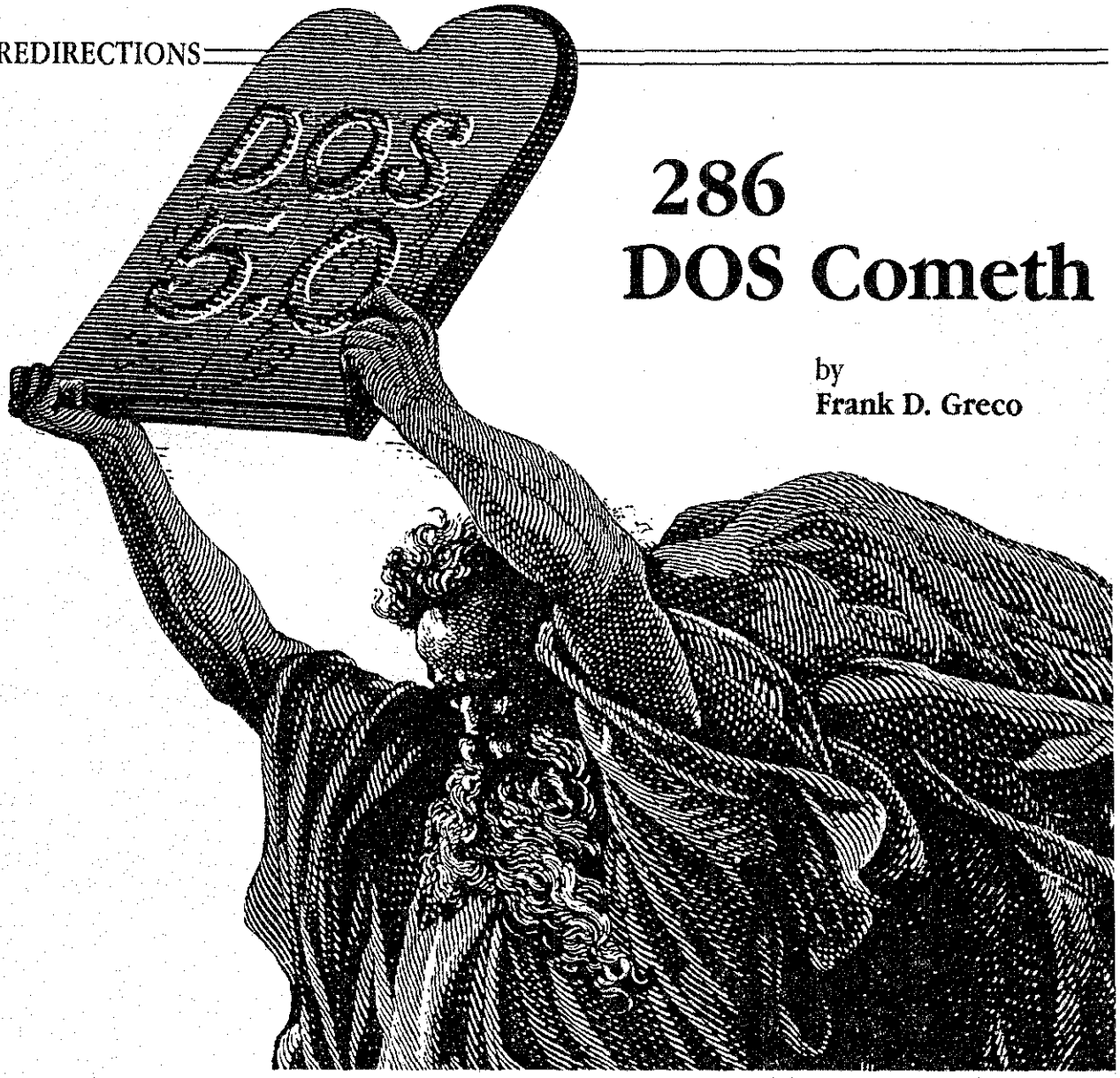
Periscope I, software, manual, protected memory board and breakout switch $295
Periscope II, software, manual, and breakout switch $145
Periscope II-X, software and manual $115

Add shipping - $3 US; $8 Canada; $24 elsewhere. Ask about air shipment if you can't wait to get your programs up and running!

PERISCOPE
The Periscope Company, Inc.
(formerly Data Base Decisions)
14 Bonnie Lane, Atlanta, GA 30328 404/256-3860

Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



286 DOS Cometh

by
Frank D. Greco

DOS Spoke the Law-ed

With the forthcoming multi-tasking (protected-mode) version of MS DOS come new restrictions on the creation of "well-behaved" programs. According to Microsoft officials at a Windows seminar I attended in New York City, programs that follow these rules (they were adamant about not calling them "guidelines") will be guaranteed of running successfully in future DOS versions and future Windows releases. I've condensed several comments from Microsoft's representatives during that seminar into a list of DOS laws. If you want to swallow these restrictions whole, here are the Ten DOS Commandments in tablet form (pun intended):

I. Thou Shalt Not Perform Arithmetic Using Segment Registers.

The 80X86 family is notoriously not blessed with a plethora of general purpose registers. To artificially mitigate this unfortunate circumstance, some system programmers occasionally use the segment registers as general purpose registers. This was never recommended, and therefore this particular restriction is understandable.

The Psychology of Segment Fraud

Undoubtedly, the segment registers will be heavily used in the multi-tasking DOS environment coming up. The "New" or "Advanced" DOS will probably make an assumption that the segment registers will point to their appropriate segments (for example, DS points to *the* data segment) when it comes time to context switch between tasks. In addition, it's a logical guess that Microsoft needs these registers to facilitate process (task) relocation and disk swapping. After all, Intel *did* intend the segment registers to hold segment addresses (pointers).

However, there happen to be several DOS functions that return values through the DS register. Will Microsoft alter these particular DOS functions to *not* return results via DS and perhaps use another method? This may break some existing code out there that assumes a result is in the DS register. For example, the DOS function 0x1B Get Default Drive Information returns, among other things, a pointer to the File Allocation Table (FAT) ID byte (an indication of how a disk is formatted) in a double word DS:BX. In their *MS DOS Programmer's Reference* manual, Microsoft recommends the following macro to perform this function:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
get_drive_info  macro
                push        ds
                mov         ah, 01Bh
                int         021h
                mov         al, byte ptr [bx]
                pop         ds
                endm
```

The macro is fine if this section of code is considered to be "atomic," that is, uninterruptable by a multi-tasking operating system kernel. The implication is that the code is safe in a single-tasking system or in a cooperative "multi-tasking" co-routine environment a la Modula II, Ada, Concurrent Euclid, or even Microsoft's Windows, where processes voluntarily give up control of the CPU.

We Interrupt This Program

However, consider what transpires relative to the section of code above in a *true*, preemptive, multi-tasking system. What if an operating system context switch, which can be initiated via a hardware timer interrupt (that is, not by a cooperating process), occurs right after the DOS system call (int 021 hex) and before the popping of DS? At that point, DS is certainly *not* pointing to the proper data segment. If this occurs on a 286, the processor's addressing mechanism does not have the "proper" selector to find the appropriate segment descriptor. It is pointing to the segment of memory in which the FAT ID is to be found. So if a context switch happens at that moment, the value of DS saved (on the stack or in a process table of some sort) due to context switching is not pointing to the "real" data area. If during the course of servicing other processes a multi-tasking operating system kernel determines to swap this particular frozen process along with its data to the disk, it could conceivably save the wrong data segment. The DOS would then lose track of the *correct* data segment and consequently, lose track of all the actual data owned by that process.

And if you're thinking about sandwiching a CLI (clear or turn off interrupt) and an STI (set or turn on interrupt) around the macro above, forget it. In protected mode these operations are inaccessible to an ordinary user process. The bottom line is that Microsoft may redesign the register interfaces to some of the DOS functions so that segment registers are not corrupted. As a matter of fact, this was alluded to during the Windows seminar. This suggests to DOS (Assembler) system programmers that it's a good idea to maintain modularity as much as possible in order to localize these potential problems so that corrections can be quickly performed. But you knew that already, didn't you?

II. Thou Shalt Not Write .COM Programs.

.COM programs are to be avoided because they are not dynamically relocatable. They are binary images that load at specified addresses and have no intrinsic relocation capability; that is, once a .COM program is loaded in memory, it is anchored there and cannot be moved elsewhere in memory. Relocation capability is neeessary in a multi-tasking environment. Microsoft seemed to suggest that .COM programs will in all probability disappear in "future" DOS and that programmers should use .EXE files instead. The reason lies in the internal structure of .EXE files. The header portion of an .EXE file contains relocation information needed for a multi-tasking environment. The header contains (among other items), for example, the initial values of the Stack Pointer (SP), Stack Segment (SS), and Code Segment (CS) and a table of items that require relocation modification before execution. The DOS loader/executer uses all this information to "set up shop" before an .EXE file can be run. This is, by the way, the reason that .COM programs load faster; they do not perform any of the preliminary setup that an .EXE file requires.

We can therefore infer that .COM-style programs, since they do not have any relocation ability, will not run under future DOS and current Windows. So we avoid .COM programs (Sigh . . . there goes my eight-byte BEEP.COM). But wait, what about programs that *must* be in .COM format—RAM resident utilities, for instance? Seek ye the next commandment.

III. Thou Shalt Not Commit SET/GET-Interrupt DOS Functions

Another surprising no-no, according to Microsoft OEM customer support engineer John C. Pollock, is the manipulation of the interrupt vectors. During the Windows seminar, Mr. Pollock said that DOS services 0x25 and 0x35 GET and SET interrupts will, in all probability, be history. Programmers will no longer be able to install their own interrupt service routines (ISRs) via a DOS call. This suggests that Microsoft wants RAM-resident software to go the way of the Nehru jacket. You can imagine the uproar this caused at the seminar. How many of you use SideKick (which Mr. Pollock said was the worst offender against DOS rules he has ever seen), ProKey, Graph-In-The-Box, DosEdit, a disk cache, printer filter, an asynchronous communications buffer, or the "user-supplied" timer interrupt? Would you like to give them up for good? I doubt it.

Mr. Pollock tried to mollify the uneasy crowd with the sop that RAM-resident utilities will still effectively be available, but in another form. He insisted that RAM-resident programs will be "replaced" by .EXE background processes that remain dormant until needed. They would be "iconified" at the bottom of the Windows screen, activated ("clicked on" via mouse) when they are needed, and deactivated ("clicked off") when the user is finished. If Microsoft does indeed eliminate the DOS facility of installable interrupt service routines, there is going to be a tremendous backlash from the user community. Mr. Pollock added, "DOS functions 25 hex and 35 hex may not be around much longer. Leave the interrupts to us."

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The inexperienced system programmer might say "So what? I'll calculate the vector address manually and poke the address of my ISR right there!" Wrongo. . . Debug-breath (apologies to Johnny Carson). The interrupt vector table, like other objects in protected mode and in the Windows environment, has no *fixed* physical address and consequently cannot be directly accessed successfully. Addresses in protected mode are "virtual" addresses; programs have no access to physical RAM addresses. As a matter of fact, the starting physical address of a load module can change during the course of its life because of process swapping. When a process is swapped back in from disk (it might originally have been swapped *to* the disk to make room for another process), it will be pure coincidence if it does start at the same physical address.

Now I See Those Pearly Gates

I find Microsoft's statements regarding the loss of user-installed interrupts a little strange because the 286 (and especially) the iAPX 386 both offer the facility to give each task its own interrupt table—in protected mode via manipulation of the IDT (Interrupt Descriptor Table) and the LDT (Local Descriptor Table). So why is Microsoft making this law about not using the GET/SET-Interrupt functions? For one thing, it forces the demise of .COM programs (Commandment II), which cannot exist in a Microsoft multi-tasking environment. The strategy apparently is that the more programs that are Windows- and "Advanced DOS"-capable, the better. If there are more Windows-capable or "Advanced DOS"-capable programs, more Microsoft products will be sold (Microsoft's Windows shell, Microsoft's Mach 10 [for millions of older PCs], Microsoft's premier spreadsheet Excel, Microsoft's TI 34010 graphics board, Microsoft's Windows-capable Language Products. . . get the picture? *Therein* lies the method to Microsoft's madness. *That's* why Bill Gates has been smiling like a Cheshire cat recently.

IV. Avoid-ith the Old IN-and-OUT Even If You Feel Like It.

Do not use the IN or OUT opcodes. This comes under the heading of "directly manipulating the hardware." Along with other consequences, this rule will certainly force all those telecommunications programs to be rewritten because everyone accesses the COM ports directly. Let's face it, the DOS COM-port serial interface is laughable and the BIOS serial interface is too slow. In addition, the ISRs for interrupts 0x0B and 0x0C (COM1 and COM2) are unbuffered (having a 2 character buffer is being essentially unbuffered!) and can lose characters if our application program is not fast enough. The common solution is to provide a programmer-supplied ISR that has a buffering strategy. Whoops. . . I just committed a sin in my thoughts. I wasn't supposed to install my own interrupts! I can see computerdom's Beelzebub waiting for my soul already.

The reason for controlling the access to external devices revolves around an old philosophy: sharing. In a multi-tasking environment, several "concurrent" processes may each require access to a resource coincidentally, and a "traffic cop" is needed to mediate the contention. If you bypass the "traffic cop" (in this case, DOS), collisions may occur—resulting in unreliable concurrent programs. Imagine the high entropy of multiple concurrent processes simultaneously sending commands to the interrupt controller, the UART, or the disk controller! The proverbial "chaos would ensue."

Programs that use the IN and OUT opcodes may not work at all in a Microsoft-based 286 protected mode environment (but will in a 386 virtual environment). In its protected mode, the 286 microprocessor offers four different protection levels or privileges. They range from highest privilege (0) to lowest privilege (3). To control access to I/O ports, the 286 processor has a 2-bit status flag called an I/O privilege level (IOPL). This flag indicates the current least-permissible privilege level in which I/O opcodes are allowed. If a process attempts to access an I/O port directly in protected mode and its privilege level is greater than the IOPL (that is, the task is less privileged), a protection exception (interrupt) will be generated. The operating system kernel can then do whatever it wants when it services the exception. The two most likely candidates for exception servicing will probably be either aborting the I/O attempt or temporarily switching to real mode to allow the process to execute (but note that switching to real mode will compromise memory and task protection!). If Microsoft abides by classical operating system design theory and only allows system calls (or perhaps DOS-internal routines) to set the IOPL to 0, concurrent user-level processes that attempt to perform direct I/O will bomb.

In order to prepare the way for the "New DOS" (sounds like a detergent doesn't it?), Microsoft suggests that programmers use the Microsoft Windows-supplied routines, which offer a rudimentary method of sharing and process privileges, to access hardware resources like ports. Do you think they'd recommend someone else's product?

V. Thou Shalt Not Violate Thine Own Virgin Instructions.

That is, no data is allowed in the code segment. Code segments have to be read-only. Presumably, this will ease the development of shared text programs, provide for reentrancy, and facilitate swapping. Microsoft representives at the seminar guaranteed that the new version (4.0) of the Microsoft C Compiler generates only "pure" (totally separate data and code segments) code. In addition, the Microsoft Assembler (MASM 4.0) has an option /P that checks your assembler code for more than 99 44/100% purity.

However, this may hinder development of some device drivers which typically store data in their code segments. And what about FORTH, Lisp, and Prolog? In these languages, code is data and data is code. It will take extra effort to port assembler-written versions of these languages to a 80286 protected mode.

VI. Thou Shalt Not Covet Physical Access to the BIOS Data and I/O Ports.

Because the physical address of objects in a protected-mode environment changes, accessing BIOS data directly (via some sort of peek( ) or poke( ) type function) is not recommended and is not likely to work consistently. We are told to use the published interrupt interfaces to the BIOS and I/O devices and not drive the hardware directly. Since direct access of the communication ports or the 8259A interrupt controller will cause contention problems in a multi-tasking environment, Microsoft offers a set of communication primitives with the Windows developers kit. Using these primitives will "guarantee upward compatibility with the 'New' Dos." I have not used them yet, but I certainly hope they did a better job than with their original DOS interface to the COM ports and IBM's "recommended" BIOS interrupt routines.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Although Microsoft is "rigorously suggesting" (that is, demanding) that only the interrupts be used to access the hardware, IBM has said that on its own "PC machines," certain I/O port addresses will remain upwardly compatible. The video retrace status ports, 8259A interrupt-control port, 8250 UART ports, and probably the game adapter port will be at their usual places for future PC machines. Be aware that IBM plays insidious semantic games, and if they release a new moniker for an advanced type of PC called, for example, the "IBM Personal Workstation," all bets are off; it wouldn't be a "PC." Also, adhering to the IBM PC hardware port specifications will not guarantee full compatibility with other PC clones, which may be built differently than IBM's PC.

Don't Rely on the Internals of the BIOS Calls

This corollary commandment (?) should be understood by all experienced system programmers. Classically speaking, the systems programmer should not be concerned with how a BIOS routine is implemented, where a particular piece of BIOS data can be found, or how the I/O ports are buffered. Most traditional systems programming deals with the interfaces, not the guts of the machine. On larger systems, programmers considered it a godsend if they could just find the documentation for interfacing to the machine's internals. Let's face it, we were all very lucky (and astonished) when IBM published the BIOS Assembler source code listings. Most of us scanned the listings hungrily looking for tips and techniques on how to program the little PC.

But we are all told that the recommended interface to the hardware is through the DOS. And as we all know, DOS (as a true operating system) is not exactly optimal or complete, so we bypass DOS and use the lower-level BIOS interfaces. In classical operating system theory, we are risking nonportability because the BIOS interfaces were never guaranteed to be the same from DOS version to DOS version. Thankfully, however, industry standards (that's read "IBM") and a strong user community are coercing the BIOS entry points to be at least consistent from version to version.

In the course of writing these faster, low-level BIOS-interfacing routines, a few of us made assumptions that certain registers, besides the registers that returned results, would be set up "just like it says" in the BIOS source listing.

These programmers and programs are going to get burnt. To implement reentrancy, which is needed for a true multi-tasking operating system, the BIOS (and DOS) routines are going to change internally. Programmers who make assumptions about the internals of a BIOS call because the BIOS source says so, are treading on thin ice. Programs that access the hardware directly will be in the same volatile position. Even my own asynchronous communications programs, which manipulate the 8259A interrupt controller and the 8250 UART chip directly with my RAM-resident communications interrupt routine, are guilty of this sin (the asynchronous communications support through DOS or the BIOS is abysmal on the naked PC). In any event, use the published recommended BIOS interfaces and not the innards of the beast for that nice, warm, cozy feeling all over.

VII. Honor Thy Video Interfaces.

One of the reasons microcomputers became so popular was their lack of system restrictions. Recall that Visicalc (and consequently the Apple II) became popular because of its instant feedback. This "feel" came from the breaking of classical operating system rules and the writing of characters directly to the screen memory. Other operating systems had no choice. For example, unless a vendor supplies a memory-mapped console and an appropriate device driver, Unix programmers must use either user-ugly, line-oriented output or slow escape-sequence-burdened output. Therefore a common complaint about most Unix systems is about the relatively slow output to the screen. Since the user feedback is slow, so is acceptance of the systems.

Please Write To Me!

As most of you experienced PC programmers know, writing to video memory directly is not exactly encouraged by Microsoft. Microsoft suggests using the DOS interfaces, or if you must, the BIOS interface, to send characters to the screen. Despite warnings from Microsoft, programmers fall prey to the speedy screen siren who promises instant screen output if you write to her. However, with a protected-mode DOS, writing directly to the screen might induce an addressing exception with the 80286 in protected mode. This would seem to suggest that all those graphics libraries that write directly to video memory for speed will have to be rewritten to take advantage of a protected-mode DOS. In most cases, the low-level routines of these libraries (if they're structured properly) will then be *forced* to call the superslow BIOS video routines. Graphics vendors will get together and send a collective prayer to the Maker above (I don't mean IBM!) to accelerate the acceptance of speedy video boards that use the powerful Texas Instruments 34010 or Intel 82786 graphics chips. But I don't see this scenario occurring until either chip is fairly well established in the market. My guess is late 1987, after the protected-mode DOS is available.

VIII. Thou Shalt Shun Memory Mapped Peripherals.

Yeah, right . . . .

IX. Thou Shalt Take the Name of Allocated Memory in Pain

Throughout the Windows seminar, Microsoft representatives repeatedly told the audience to start using the Windows-supplied memory management routines for memory allocation and deallocation. The hint was that these routines are "very similar" to the mechanisms of the "New DOS."

Microsoft's main systems software man Steve Ballmer emphasized that "Current DOS memory management is rudimentary. We are addressing this in Windows . . . and in the future versions of DOS." During another talk, John Pollock said, "Windows uses the 286 MMU philosophy. . . read chapter 5 of Morse's book." He was referring of course to *The 80286 Architecture* by Stephen P. Morse and Douglas J. Albert. I would second the motion; anyone who is doing serious systems software development with the 286 should read the entire Morse book. Surprisingly, Mr. Pollock also said that programmers should avoid writing C programs that use standard library memory management routines such as malloc( ) and free( ) because they will *not* even work in a Windows environment! In addition, programs that use the C standard library output families fprintf( ) and fputs( ) will also not work correctly in Windows! In rather fascistic tones, Mr. Pollock said, "You must use the Windows-supplied routines to do these things."

This is precisely what I don't like about Windows. You must conform to their rules. Older programs that do use malloc( ),

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

printf( ), etc., can't run comfortably in a Windows environment (that is, in a multi-tasking window). Wouldn't it be nice if Windows were developed as a set of device drivers and incorporated into the DOS logical file system? That implies that all file-related system and library calls—fopen( ), fprintf( ), read( ), etc.—would also refer to windows. To perform popup menus and mouse tracking a series of ioctl( )'s would be used. Instantly, all languages would have access to a simple Windows environment. You would be able to do things like:

```
      BASIC
      :
open "win1" for output as #1            ' Open requested window
print #1, "This is going to window 1"   ' Simple, isn't it?
      :



      C Language
      :
if ( (fh1 = open("win1", O_RDONLY)) == -1 )
      error("Can't open window 1");
else
      num = read(fh1, buffer, 80);      /* Read from window 1 */
      :
      :
if ( (fh2 = open("win2", O_WRONLY)) == -1 )
      error("Can't open window 2");
else
      write(fh2, buffer, num);          /* Write to window 2 */
      :
```

Besides that, file and command redirection would be accessible to programs that write to Windows. That means older Windows-ignorant programs that read and write to the screen via the file handle method can do so in a window. A front end process would set up the file handles accordingly (that is, redirect stdin and stdout to Windows) and then merely exec( ), that is, overlay, the older process, which uses the altered file handles and is consequently window-able. However, from a marketing point of view, this obvious flexibility might cost Microsoft. Anyone could then implement the drivers and, if they did a superior job, take the business away from Microsoft. Aye there's the rub; the IBM way of thinking has rubbed off on Microsoft. In mainframe fashion, programs that are rewritten for the Windows environment can run only in that environment.

For a truly flexible and elegant example of what a window environment should be, investigate Sun Microsystems' new Network Extensible Window System (NeWS). It was formerly called SunDew and is based on the PostScript graphics language. When I asked Microsoft's John Butler, who struck me as a real classy and insightful guy, to compare Sun's SunDew (a.k.a. NeWS) and Microsoft's Windows, John said: "Sun Microsystems is supposed to be in front of us [Microsoft]. . . that's their job."

X. I AM the Law (by God!). Thou Shalt Not Have False Gods Before Me

No doubt some other vendors of systems software will try to persuade programmers and end-users to avoid Microsoft's Windows and multi-tasking MS DOS. IBM, for one, believes that the 80386, not the 80286, is going to be the premier microprocessor for the business world for years to come. A major operating system for the 386 may be the cornerstone of IBM's strategy for the future. And I would bet that the VM-like Virtual Resource Manager (VRM), currently a part of the Unix-based IBM PC RT, figures in that game plan.

Consider this scenario: In its usual style, Big Blue waits for Microsoft to make the first move in the multi-tasking chess game. An 80286 protected mode multi-tasking kernel is Mean Green's predictable opening move. At this point, several 286 computer vendors have already announced their 386 machines; they're merely very quick PC's. IBM counters with a 386 version of multi-tasking MS DOS with a flexible graphical interface called "Advanced MS DOS" or "MS DOS It" or "No TopView. . . Phew," which unlike Microsoft's DOS, will run older DOS programs concurrently with new ones and subsequently closes the Gates. It's not a likely scenario, but an interesting one.

I only contemplate this improbable scenario; I really don't think Microsoft is worried. They're currently trading around $40 a share in a depressed industry, and their Macintosh business is booming (rumors of a PageMaker-like program abound around the Redmond Rogues). But why did Bill Gates say (quoted in a popular technical publication), ". . . stupid virtual monitors with their stupid user interfaces. . ." when he was asked about his competition? Sounds to me like someone concerned.

Conclusions and More Rambling

I Wanna Be Just Like Dad

Like the Unix operating system, the multi-tasking version of DOS is being written in the C programming language. Because of this, the future DOS will be fairly portable to non-Intel processors. As I mentioned in "Redirections" back in the early part of 1985 (*PJ*, Volume 3, Number 2), having MS DOS also available on non-Intel processors would make it as portable as Unix.

Other major microprocessor vendors such as Motorola, National Semiconductor, and Fairchild Semiconductor (now a Hitachi affiliate) are all hoping that since applications will have to be rewritten to use the advanced features of the new portable DOS, the programmers could be directed to use a standard interface to DOS. Already companies like Hunter Systems in Palo Alto are working with the microprocessor manufacturers to try to establish a standard for a C language interface with an advanced DOS to be called XDOS (BIOS and video memory standards are also being drafted). If programmers abide by this standard (be it defacto or formal), 68020s, 32032s, and Clippers will then run advanced MS DOS programs and hence be legitimate MS DOS engines. And since the New DOS would be functionally equivalent to Unix, perhaps it may be tweaked just enough to conform to the upcoming POSIX and Unix System V standards, which uncoincidentally, opens the door to potential government business and Manufacturing Automation Protocol (MAP) network bids. The more I think about it, the more intriguing it gets.

Microsoft Is Their A's

Judging from the reactions of Microsoft's representatives to questions relating to porting Windows or a multi-tasking DOS to the 680X0 family, I would say it's definitely in the works. One technical representative told me during a break: "I cannot comment, but what would you think. . .with Atari, Apple, and Amiga [Commodore-FG] controlling the lucrative home market." And during lunch, Microsoft's John Butler told me: "The 68K rules the home. . .when we get high quality proportionally spaced text on screen, a whole new class of users will 'find' computer technology. . . .We [the computer industry] have a responsibility to our fellow man, the one with normal human CPU amperages, to provide these tools." Anyone want to bet on a 680X0 port?

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Felix and Oscar

And if you're wondering about using Windows on top of Xenix, consider this quote from Microsoft: "The Xenix implementation of Windows has been considered, but we have not really started there. Multi-tasking DOS has higher precedence over Xenix issues." In my opinion, the development of a Xenix version of Windows is not commercially warranted yet and would drain too many valuable resources at Microsoft. Besides, Microsoft has 60 Unix-based Sun machines connected to their "on-campus" network; I'm sure they are very impressed with Sun's NeWS window software. In comparision, NeWS makes Windows look as sophicticated as EDLIN.COM.

Now, Betcha Can't Do This!

The classical operating system restrictions that Windows and the New DOS require are new to the microcomputer world. They have been recognized and dealt with for years by the mainframe and minicomputer industry. It will take a while for programmers to get used to a protected mode environment. As Microsoft's Mr. Pollock says, "Windows is like being in jail. You must request the operating system to do anything."

Remember how microcomputer programmers used to taunt their minicomputer and mainframe counterparts by describing all the hacks they did to their own *personal* machines? How interrupts, system subroutines, or low-level data were altered at will to do this spiffy thing or that cutesy thing? Microcomputer programmers had freedom, carte blanche, to do what they wanted. Is it truly necessary to have tremendous memory protection among cooperating user processes on a single-user windowed workstation all the time? Perhaps what is mostly needed is large memory addressability and "friendly," cooperative multi-tasking. For many vertical applications, cooperative co-routining mechanisms, sometimes called "internal processes" (per Brinch Hansen), "lightweight" (Sun Microsystems), "inexpensive," or "non-kernal," processes, should suffice. Coupled with shared memory, where all tasks share the same memory, co-routines solve many multi-tasking applications.

So I'm going to predict that there will be a branching off the DOS family tree: a protected mode DOS and an unprotected mode DOS with extended addressing. One would be a safe, commercial, applications-oriented, police-state version; the other a vertical market, take-over-the-interrupts, systems programmer's playground. An unprotected mode, large addressability DOS with co-routine-style multi-tasking would be especially inviting for the 386 with 32 bits (4 gigabytes) of linear addressing. Windows, the descendent of the Viewers package of Xerox's Cedar environment, would then be optional for both versions.

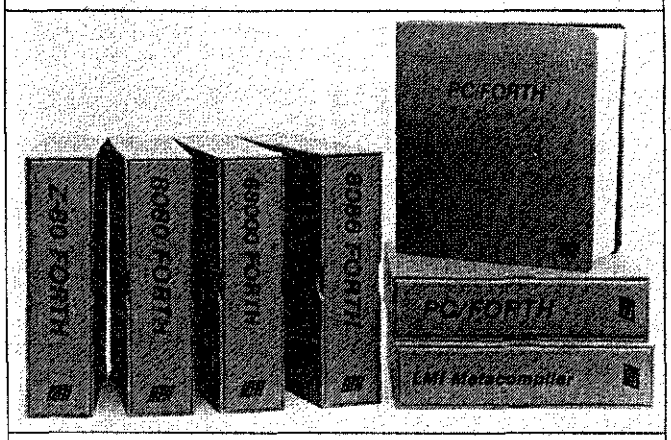
When the protected-mode, multi-tasking MS DOS arrives, a lot of microcomputer programmers will be surprised at the new OS's restrictions. They'll find out that it's going to be a matter of what's possible vs. what's permissible. □

Frank D. Greco is an independent consultant who specializes in highly usable computer systems.

Readers can contact Frank at the following address:

Frank D. Greco
P.O. Box 470
Kearny, NJ 07032-470
(201) 754-7820

TOTAL CONTROL
with *LMI FORTH*™



For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler
- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Number 63 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Marketing and Rules to Live By

by
Paul Barkley

Marketing Yourself through Contacts
If the first duty of a politician is to get elected, then the first duty of a consultant is to find clients. This is either (a) no problem (your telephone is ringing off the hook) or (b) a serious problem (there are only two cards in your rollodex, and one of them is your mother's).

Finding new clients is a difficult task because they're hard to separate from the mass of disinterested businesses. It's just as hard for an interested potential client to locate *you*, though.

Do you advertise in the yellow pages? That costs too much for many independent consultants, but do you at least take your free listing under the heading of your choice? Sure you do, but what heading do you pick?

Each area's yellow pages is a little different, but there are usually a dozen computer headings in a major book. Even if you know the difference between "Data Processing—Consultants and Designers" and "Data Processing Service" and between "Computers—Software and Services" and "Computers—System Designers and Consultants," it's highly unlikely that your prospective clients will. Most of them will wind up lost under "Computer Dealers," looking through page after page of ads.

Only a very dedicated reader will make it through the computer and data processing headings all the way to "Dating Service," where it's easy to get distracted. And then it's still impossible to tell from the one-line listings who does what (we're back to computers here, forget the dating service), regardless of the heading. As they're currently structured, the yellow pages don't work for consultants. Period. I don't know about dating services.

A better marketing strategy is to become a member of a consultants' organization that publishes its own directory. The Independent Computer Consultant's Association (ICCA) in St. Louis has chapters around the country, and some of the local chapters (like WICCA in Washington, DC) publish directories. National listings are not usually useful unless you're nationally known, in which case you don't need them, so look for a local group.

The biggest problem with directories is getting them distributed. If they go out only to the members, they won't do you as much good as they should. Although there's a great deal of talk about sharing business among fellow members, practical matters make that somewhat difficult.

First of all, one member must know another fairly well to recommend him or her to a client. Other members must know what specialty you provide and be able to recall it instantly. If you and twenty other people all work in dBASE, no one will remember you. The one architect and CAD specialist in an organization, on the other hand, may be well remembered even if the demand for his or her services is smaller.

Secondly, how likely is it that one member will get an unsolicited request for your specialty? Is this member better linked to a network of potential clients than you are? Does one of this member's clients now need your particular specialty that your fellow member can't provide? Perhaps. Most consultants, though, operate within a very small group of client companies, and their contacts may not be that extensive even if they're billing 40 hours a week, year after year.

Third, I've seen a number of aborted joint efforts, where two independents tried to team up to offer a prospective client combined skills. The usual problem is that since the two are not really part of the same company, the client gets suspicious when only one person has business cards, and both people have different office phone numbers and/or addresses. Also, it's not easy to divide the pie. I find these arrangements generally a waste of my time.

Subcontracted work, on the other hand, doesn't seem to be too popular among independents because one consultant cannot generally afford another consultant. Why pay your peer what you pay yourself? Why not find some way to slide back the deadlines so that you can do the work alone? Many consultants worry about slack time and don't want to "off-load" any work if they can help it. Marking up another consultant's work under your umbrella usually makes that work too expensive for your clients, by definition, since they're used to paying your rates. Rarely can you convince a client who may already be nervous about your rates that you need another, more expensive, expert to assist you.

Thus, what you need is a directory primarily intended for clients, not your peers.

WICCA made an effort to attend several local business/computer shows and through their booth distributed a number of copies of their directory. This resulted directly in business for me, and I'm sure for other members as well. I learned that the directory had floated around one very large organization until a chance hall conversation caused it to change hands. It now sits on my contact's desk and was the *only* way she would ever have located me. She didn't attend the show, but the resource material was available within her company.

So, I recommend that you join a local group and make a concerted effort to get

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

a directory out to the right people. If you don't want to join a group in your area because they're all turkeys and you're Joe Cool, think of it as a professional challenge. You need the exposure if your consulting business is to survive, so plan on spending the required hours per month working to change the organization—don't wait for others to do your work for you.

I log all my billable and unbillable hours so that I'm able to see how much time is spent on various activities, and you can bet that all the hours put into organizations are logged against marketing. I find that tracking the hours spent in groups helps reinforce the importance of these activities and makes it easier to face the prospect of making a bunch of calls to locate a meeting place or answering questions about the organization from prospective members who want to talk all afternoon. Marketing is just another form of work and must be done even if you're a "techie" with no heart for this sort of thing. Do it.

A useful strategy for some people is to become an officer of each group to which they belong. They find that it greatly increases their visibility and leads to lasting peer contacts that eventually pay off in referrals.

This is based on the networking concept. You must get to know other people if you expect to be able to share information, contacts, and referrals. At every meeting that I attend, I find quite a few people who want to "hang around" but have no idea how to socialize. They gravitate to other members who are talking in small groups, but have difficulty "breaking in." They don't know anyone and feel out of place. Since many older members have forgotten not only why they attend these meetings (for work, not just play), but forgotten their manners as well, it's common to see new people drifting from group to group without getting a word in edgewise. Once you get established, you should make the effort to introduce yourself to people who drift up to a conversation you're in. It's disruptive sometimes, but you never know who you'll meet that way.

If you're a new member and don't know anyone, you'll have to try to break into a conversation when you can find something (intelligent) to add. If you have experience with a very new data base, for example, you may be astounded at how eager other members are to find out what you know and to exchange information.

By all means, learn how to graciously exchange cards. The purpose of being there, other than perhaps picking up some technical information from the presentation, is to network. If you don't give out your card and get other people's cards, you will have wasted your evening. (If you arrive at a meeting without cards, you should simply go over to a corner and drink yourself into a stupor because you can't possibly make a worse impression than by not having a business card.) Even if you haven't formed a company, at least get some cards printed with your name, address, and phone number and perhaps a line about consulting. The ten dollars will be well spent.

Many people are awkward about exchanging cards. Try responding to what someone says with, "That's very interesting. Do you have a card with you?" When you receive the card, give him or her your card. If you're talking with a prospective client instead of your peers, you might offer to send that person some additional information about the topic under discussion—this way the prospective buyer doesn't feel pressured into giving out a card, and you open a line of communication.

For social reasons, dinner meetings are popular with some groups because they give members an excuse to be doing something while they're trying to get acquainted. When you run out of clever things to say, you can ask that the butter be passed.

At these meetings I chafe while listening to stories about car repair, surgical procedures, wine tasting trips to Europe, and other non-computer activities, but these are all part of establishing relationships. It's often difficult to carry an entire evening on technical matters, and personal anecdotes help form relationships. To tell you the truth, I really dislike attending meetings and wouldn't do it if it weren't so important. I remind myself that spending a couple hours meeting people over dinner is a lot better than spending a couple hours with new clients who have no idea what they want. The former gets me fed, at least, while the latter only gives me a headache.

Finally, the most important thing to remember about joining groups is that they never pay off in the short run. Many people make the mistake of dropping out of a group after three or four months when it appears that nothing is happening. Give it time. You have to keep going to meetings month after month for at least a year before you can expect to see much in the way of concrete results. Many people's consulting careers don't last long enough to see the fruits of this association.

Rules to Live by

I'd like to hear from people who have little rules that they find enhance their consulting and programming efforts. I have one

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMBINE THE
RAW POWER OF FORTH
WITH THE CONVENIENCE
OF CONVENTIONAL LANGUAGES

HS/FORTH

Why HS/FORTH? Not for speed alone, although it is twice as fast as other full memory Forths, with near assembly language performance when optimized. Not even because it gives MANY more functions per byte than any other Forth. Not because you can run all DOS commands plus COM and EXE programs from within HS/FORTH. Not because you can single step, trace, decompile & dissassemble. Not for the complete syntax checking 8086/8087/80186 assembler & optimizer. Nor for the fast 9 digit software floating point or lightning 18 digit 8087 math pack. Not for the half megabyte LINEAR address space for quick access arrays. Not for complete music, sound effects & graphics support. Nor the efficient string functions. Not for unrivaled disk flexibility — including traditional Forth screens (sectored or in files) or free format files, all with full screen editors. Not even because I/O is as easy, but far more powerful, than even Basic. Just redirect the character input and/ or output stream anywhere — display, keyboard, printer or com port, file, or even a memory buffer. You could even transfer control of your entire computer to a terminal thousands of miles away with a simple >COM <COM pair. Even though a few of these reasons might be sufficient, the real reason is that we don't avoid the objections to Forth — WE ELIMINATE THEM!

Public domain products may be cheap; but your time isn't. Don't shortchange yourself. Use the best. Use it now!

HS/FORTH, complete system: $395. with "FORTH: A Text & Reference" by Kelly and Spies, Prentice-Hall and "The HS/FORTH Supplement" by Kelly and Callahan

VISA Visa Mastercard MasterCard

HARVARD
SOFTWORKS
PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390

Number 42 on the Reader Service Card

that I'd like to share with you.

Rule: Never program after 10:00 p.m. because all the world's great computer disasters happen between ten and midnight.

Depending on the hours you keep, you may want to customize this rule to suit yourself. Certainly programmers are legendary for working through the wee hours of the night, but to a person who likes at least two meals a day and eight hours of sleep, this lost its appeal many years ago. Not only do I get grouchy and unproductive, but the big mistakes always happen when I'm tired and not paying close enough attention.

I broke the rule this summer, and once again I paid the price. I returned to the office to spend the evening working on some personal database activities that I recently began. I had this consuming idea of how to organize vast quantities of genealogical information that had gotten completely out of hand, and I continued to create new reports and enter several hours' worth of data to prove the viability of the system. (Of course a good systems analyst doesn't design his own systems the way he does his clients'—he just wings it, right?)

The air conditioning in my office building goes off in the evening, and the office was sweltering, but in my delirium I continued shoving all sorts of unrelated facts into dBASE, gleefully anticipating the first revelations to emerge as data was compared and organized.

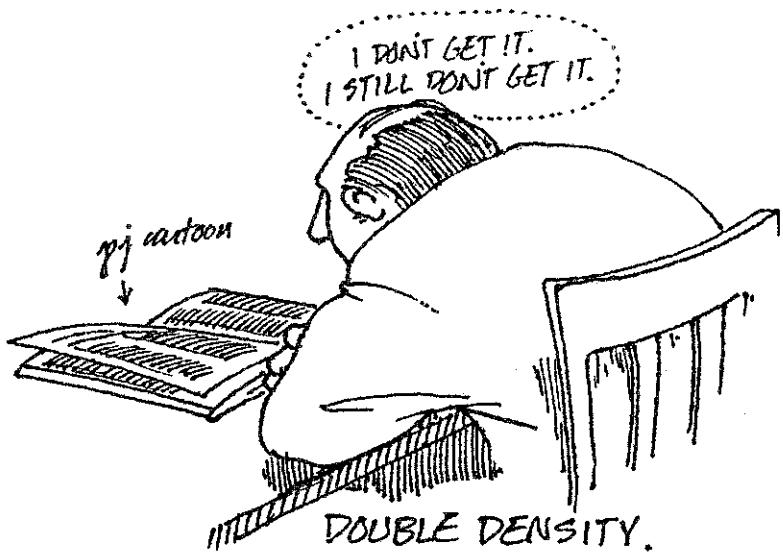
The temperature continued to rise (it was July), and at 10:45 p.m. I had all my test data in. "Let's see, should I now back all of this up or wait until tomorrow?" I mused. Then an idea for a new report format appeared as if by magic, and I couldn't wait to prove or disprove conclusively that I was descended from a long line of hobbits starting with Bilbo Baggins! Just a few more keystrokes and. . .

Garbage all over the screen. dBASE III had suffered a total nervous breakdown and echoed junk. It wouldn't respond to threats, curses, escape codes, or control codes. My system probably learned the awful genealogical truth and was keeping it from me, but in the meantime a reboot showed that 90% of my data was gone—about 8 hours' worth. (I was able to duplicate this feat the following week—this time with backups—by sending data to the printer that wasn't turned on, so it had nothing to do with the heat at all. The only thing the heat affected was my brain.)

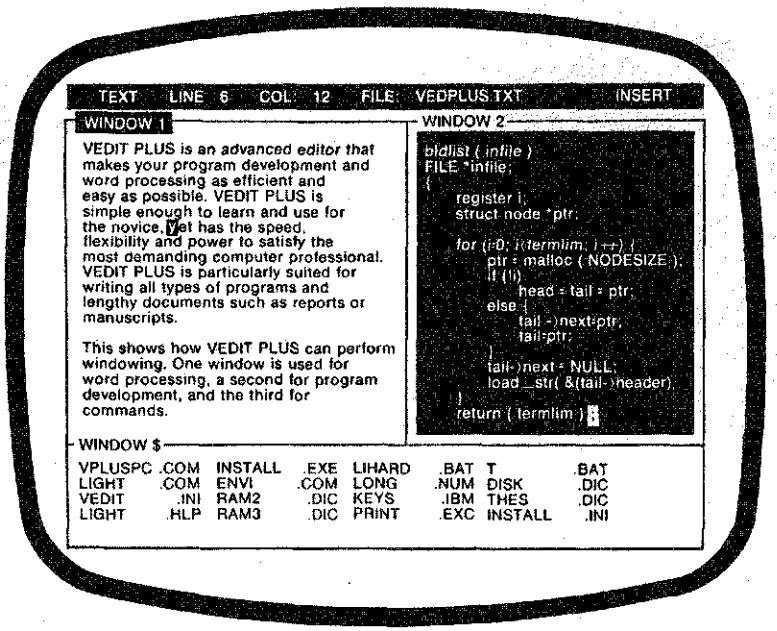
Will I now practice the backup procedures that I drill into my clients? Have I learned anything? Sure—I learned once again not to write code or enter data after ten o'clock. Like Cinderella and her transportation system, my computer system (and my brain) are temporally lycanthropic—they turn into were-vegetables every evening at ten. Everybody knows a rutabaga can't add. □

Send your thoughts, rules to program by, questions, or comments about consulting to

Paul Barkley
The Barbican Company
6845 Elm Street
Penthouse One
McLean, Virginia 22101
CompuServe: 74776, 2526



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED


TEXT LINE 6 COL 12 FILE: VEDPLUS.TXT INSERT
WINDOW 1
VEDIT PLUS is an advanced editor that makes your program development and word processing as efficient and easy as possible. VEDIT PLUS is simple enough to learn and use for the novice, yet has the speed, flexibility and power to satisfy the most demanding computer professional. VEDIT PLUS is particularly suited for writing all types of programs and lengthy documents such as reports or manuscripts.
This shows how VEDIT PLUS can perform windowing. One window is used for word processing, a second for program development, and the third for commands.
WINDOW 2
bldlist (infile)
FILE *infile;
{
register i;
struct node *ptr;
for (i=0; i<termlim; i++) {
ptr = malloc (NODESIZE);
if (!i)
head = tail = ptr;
else {
tail->next=ptr;
tail=ptr;
}
tail->next = NULL;
load_str(&(tail->header);
}
return (termlim);
}
WINDOW $
VPLUSPC .COM INSTALL .EXE LIHARD .BAT T .BAT
LIGHT .COM ENVI .COM LONG .NUM DISK .DIC
VEDIT .INI RAM2 .DIC KEYS .IBM THES .DIC
LIGHT .HLP RAM3 .DIC PRINT .EXC INSTALL .INI

VEDIT PLUS

#1 CHOICE IN PROGRAMMABLE EDITORS

VEDIT PLUS has been the #1 choice of professionals since 1980. Our latest release is even better - you can open windows to simultaneously edit several files, access many editing functions with pop-up menus, use keystroke macros to speed editing, and run other programs or DOS commands from within VEDIT PLUS.

Whether your needs are program development, technical writing or word processing, VEDIT PLUS is your answer. With over 40,000 users you can depend on VEDIT PLUS to perform consistently and reliably. It is simple enough to learn for the novice, yet has the speed, flexibility and power to satisfy the most demanding professional.

Compare. No other editor is as powerful - unlimited keystroke macros, instant 'off-the-cuff' command macros utilizing a complete programming language, single command file comparison, special word processing and programming features. No other editor is as easy to use - on-line help, pop-up menus, 75 page tutorial, 380 page manual, and VEDIT PLUS is completely customizable.

Fully supports color windows on IBM CGA & EGA, and even windows on most CRT terminals (including CRT's connected to an IBM PC). Available for IBM PC, TI Professional, Tandy 2000, DEC Rainbow, Wang PC, MS-DOS, CP/M-86 and CP/M-80. Order direct or from your dealer. $185

"To sum things up, VEDIT PLUS is a small, fast, sophisticated editor with a wealth of features and a good macro language. It offers many rewards for the dedicated programmer."
Computer Language, Chris Wolf, Scott Lewis, Mark Gayman 6/86

"VEDIT PLUS is a wholly remarkable program: blindingly fast, extremely powerful, and highly flexible."
Profiles Magazine, Robert Lavenda 4/86

VEDIT PLUS FEATURES

- Simultaneously edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname and CP/M user number support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 text registers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size, any CRT.
- Optimized for IBM PC/XT/AT. Also 132 column & up to 70 lines.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I or PASCAL.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert WordStar and mainframe files.
- Print any portion of file; separate printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts keyboard input, 17 bit algebraic expressions, variables.
- CRT emulation within windows, Forms entry.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Main menu

VEDIT and CompuView are registered trademarks of CompuView Products, Inc. MS-DOS is a registered trademark of Microsoft. CP/M is a registered trademark of Digital Research. WordStar is a registered trademark of MicroPro.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

Number 56 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Redirecton, or "They Went That-a-way"

by
Frank D. Greco

Besides being a good name for a column, redirection is the ability given to a process to change its source or destination I/O data streams from one file or device to another. Most of us have already used redirection on MS DOS or Unix systems in the form:

MS DOS	Unix
`C:\> dir >dirfile`	`$ ls >lsfile`

In both of these examples, the output of the dir and ls commands, instead of going to the default output device, was set up to be redirected into a file by the respective shells COMMAND.COM and the Bourne Shell.

As far as I know, the technique of altering the default data streams originated with the MULTICS operating system in the mid-60s. It certainly was popularized by the Unix operating system. As opposed to popular (that's read "end-user") opinion, Unix does *not* implement redirection; it merely facilitates its implementation. The Unix shell (and some other user-level utilities) really performs the redirection. In like fashion, it turns out, MS DOS does not directly implement redirection. It merely offers similar Unix-like mechanisms to processes to alter their data streams.

Additionally, it is a common fallacy to say, "MS DOS can redirect only the standard input and standard output data streams and not the standard error." It is COMMAND.COM, not MS DOS, that unfortunately restricts the user (programmer?) to stdin and stdout redirection.

Let's look at the details of how redirection is specifically done on MS DOS machines.

Getting a Handle

First of all, the recommended way to use files after DOS 2.0 was released was through the use of file handles. File handles, called file descriptors in the Unix domain, are small non-negative integers that refer to previously opened files. They are obtained via a call to the DOS open-handle system call. The open( ) function that most of you C programmers know is actually an interface routine to this open-handle system call, DOS interrupt 0x21 function 0x3D. So from the C level, a file handle is obtained in the following manner:

```
        :
int fh;
        :
if ( (fh = open("filea", mode)) != -1 )
{
     fputs("Cannot open file.", stderr);
     exit(CANT_OPEN);
}
printf("The file handle for 'filea' is %d\n", fh);
        :
```

Once the file is successfully opened, the programmer no longer uses the file's name to refer to it; he/she uses the file handle. Since DOS carries on the Unix tradition of logical file/device I/O, a programmer could open the printer or communications port just as easily as a file. Unfortunately, this does not apply as cleanly to directories as it does in Unix; perhaps newer versions of DOS will correct this anomaly.

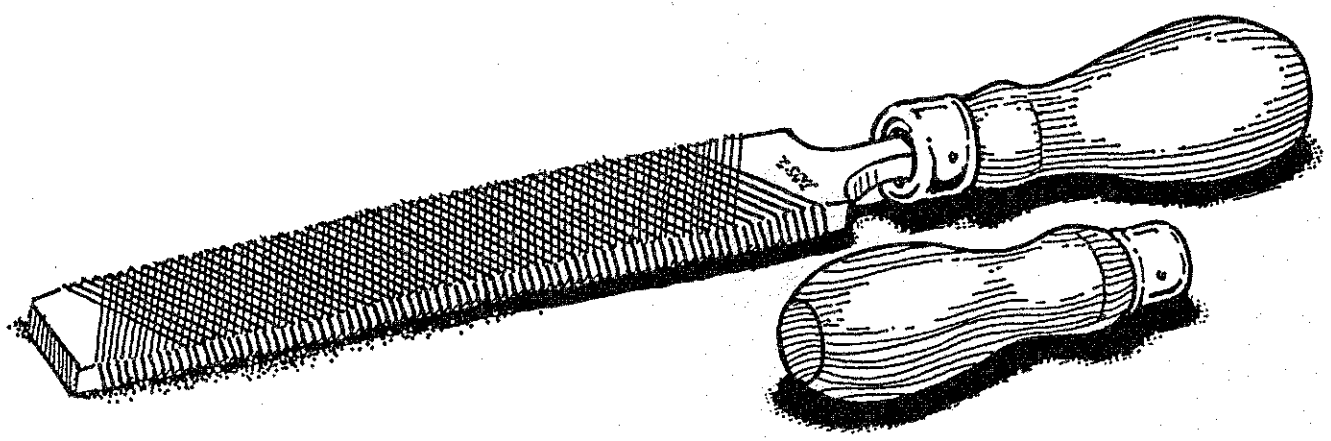
Just for the record, "opening" a file means that the operating system has sufficient knowledge concerning the file in order to manipulate it. Internal data structures in the DOS are allocated if needed and initialized to appropriate values. So if the operating system refuses to open a file, you can logically guess that there's a problem reading the disk (disk media error) or with the internal data structures (exceeded open file limit, access mode error, or the OS just could not find the file).

Getting Handles From Pointers (or was it the other way around?)

It turns out that DOS keeps a table of these little file handles in the per-process Program Segment Prefix (PSP). Currently, since each MS DOS (2.0 and 3.0) process can open a maximum of 20 files per process, the file handle table in the PSP is 20 bytes long. Assuming that DOS is internally very Unix-like (I know for a fact that some of Microsoft's DOS developers were weaned on Unix), the file handles are presumably offsets into another DOS-internal table that contains the associated DOS file-related data structures.

Be careful that you don't confuse file handles with file pointers. File pointers are obtained primarily from the standard I/O library call "fopen( )." They are the addresses of user-level (that is, non-kernel) data structures. These are higher level and more portable entities. If you look at the standard I/O header file (stdio.h), you will see a data structure that looks like:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



```
/*  Paraphrased from the Microsoft C 4.0 compiler  */

     #define   _NFILE     20
     #define   FILE       struct _iobuf
               :
     extern FILE
     {
          char *_ptr;     /* Ptr to char in buffer to be read or written. */
          int _cnt;       /* Number of chars left in buffer. */
          char *_base;    /*   I'm not sure about this one...  */
          char _flag;     /* Set by DOS I/O on EOF or ERROR */
          char _file;     /* File handle for this pointer */
     } _iob[_NFILE];
               :
               :
     #define   stdin      (&_iob[0])
     #define   stdout     (&_iob[1])
     #define   stderr     (&_iob[2])
     #define   stdaux     (&_iob[3])
     #define   stdprn     (&_iob[4])
```

There are two things to note here. First, this header file is specific to Microsoft's C compiler under DOS; other compiler writers (Unix or DOS) may decide to change the macro and variable names. In any case, your compiler should implement it in a very similar fashion. Secondly, __iob[ ] is an array size 20 (that is, __NFILE) of type FILE (not really, since FILE is a macro for "struct __iobuf"). The macros stdin, stdout, stderr, stdaux, and stdprn are essentially file pointers (FILE *). So therefore, a file pointer is not a file handle; it is the address of a particular element of the __iob[ ] array (for example, &__iob[whatever]). However, a file pointer can help you find its associated file handle. One of the structure members of struct __iobuf called "__file" is the value of the associated file handle. The macro fileno( ), typically found in stdio.h, is often used to obtain a file pointer's associated file handle:

```
#define   fileno(fileptr)     ( (fileptr)->_file )
```

Remember, it is this little non-negative integer, the file handle, that is the basis of I/O redirection.

Keep Your Handles on the Table Cowpoke

Now that you understand the association between pointers and file handles, let's return to that 20-element file handle table in the PSP, which we'll call filehandle[ ]. It is located at offset 0x18 of a process's PSP. If we inspected this table, we would see (assuming no redirection indicated on the command line):

Filehandle[] Table

Element:	0	1	2	3	4	5	6	etc.
FH Value:	1	1	1	0	2	0xFF	0xFF	etc.
Implies:	stdin	stdout	stderr	stdaux	stdprn	N/A	N/A	etc.

The first five elements (of 20) in the filehandle table, elements 0 through 4, have usable file handles. COMMAND.COM opens these 5 files (stdin, stdout, stderr, stdaux, and stdprn) when it is invoked. The other filehandle table elements contain 0xFF, which indicates that this element is unused and ready to refer to an opened file. Notice that stdin, stdout, and stderr all refer to the same file handle (1), which happens to be the CON: device.

If you are observant, you can correctly infer that if the values in the filehandle table change (for example, filehandle[1] becomes a 3), the destination or source of the data stream is altered, or "redirected." This is the key to the entire redirection mechanism.

Handle Breeding

Some other vital information deals with subprocess spawning. Any process spawned from another inherits all the open files of the original process. The legacy of the parent process to all of its child processes, as they are called, is bequeathed in terms of inherited file handles. So if a parent process wanted to redirect the previously defined data streams of a child process, it would set things up properly before allowing the child to begin playing. This is how COMMAND.COM works. When redirection is detected on the command line via '>,' '<,' or '>>,' COMMAND.COM changes the file handle table appropriately and spawns the child who inherits the table values. Notice that the child process is oblivious to the changes made by the parent. For example, if the child calls printf( ), the output will be sent to the file handle associated with stdout.

To demonstrate this concept, assume we have a program that writes to both stdout and stderr:

```
/*
 *   outpgm.c - Output to stdout and stderr I/O streams
 */
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BUSINESS PROGRAMMERS

TURBO BUSINESS ACCOUNTING MODULES

CREATE CUSTOMIZED VERTICAL ACCOUNTING MARKETS

FIVE SEPARATE MODULES
- General Ledger
- Inventory
- Accounts Payables
- Cost Accounting
- Accounts Receivables

STANDARD WITH EACH MODULE
- Full Documentation
- Executable File
- Documented Pretty Printed Source Code

ALL INTEGRATABLE **NOT COPY PROTECTED**

NO ROYALTY FEES

All for the price of $49.95 per module. Get three modules for the price of two at $98.95. Or the complete set which includes all five for $199.95. Demo disk available $10 applied to purchase of modules.

(Note: Each module comes on a DSDD 5¼ disk)

SYSTEM REQUIREMENTS: All of the above modules require TURBO PASCAL 3.0 or BETTER, TURBO DATABASE TOOLBOX, 256K, and an IBM PC/XT/AT or MS-DOS Compatible.

Call or write for full information.

PS BUSINESS SYSTEMS
4400 Wallingford Ave. N. Suite 3
Seattle, WA 98103 U.S.A.
(206) 632-4568
COMPUSERVE 73267,1513

CHECK, MONEY ORDER OR COD IN U.S. FUNDS

SHIPPING: $5.00 U.S., $7.50 CANADA, $10.00 Caribbean, Hawaii $20.00 Overseas Air.

Turbo Pascal and Turbo Database Toolbox are registered trademarks of Borland International, Inc.; MS-DOS are trademarks of Microsoft Corporation; IMB, PC, XT, and AT are registered trademarks of International Business Machine Corp.

Number 82 on the Reader Service Card

FORTRAN PROGRAMMERS

Looking for the right PC FORTRAN LANGUAGE SYSTEM? If you're serious about your FORTRAN programming then you should be using F77L- LAHEY FORTRAN.

Editor's Choice - PC Magazine

- Full FORTRAN 77 Standard (F77L is not a subset)
- Popular Extensions for easy porting of minicomputer and mainframe applications
- COMPLEX*16, LOGICAL*1 and INTEGER*2
- Recursion - allocates local variables on the stack
- IEEE - Standard Floating Point Arithmetic
- IMPLICIT NONE
- Long variable names - 31 characters
- Fast Compile - Increase your productivity
- Source on Line Debugger (Advanced features without recompiling)
- Arrays and Common Blocks greater than 64K
- Clear and Precise English Diagnostics
- Compatibility with Popular 3rd Party Software (i.e. Lattice C)
- Easy to use manual
- Technical Support from LCS

F77L - THE PROGRAMMER'S FORTRAN

$477.00 U.S.

System Requirements: MS-DOS or PC-DOS, 256K, math coprocessor (8087/80287)

FOR MORE INFORMATION: (702) 831-2500

Lahey Computer Systems Inc.
P.O. Box 6091 Incline Village, NV 89450/U.S.A.

International Dealers:

England:	Grey Matter Ltd.,	Tel: (0364) 53499
Denmark:	Ravenholm Computing,	Tel: (02) 887249
Australia:	Computer Transitions	Tel: (03) 537-2786
Japan:	Microsoftware, Inc.,	Tel: (03) 813-8222

SERVING THE FORTRAN COMMUNITY SINCE 1967

Number 44 on the Reader Service Card


```
#include <stdio.h>      /* Needed for stdout and stderr macros */

main()
{
     fprintf(stdout, "I'm being sent to stdout.\n");
     fprintf(stderr, "I'm being sent to stderr.\n");
}
```

Since both stdout and stderr are logically connected to the console, we get the following as output:

```
C:\> outpgm
I'm being sent to stdout.
I'm being sent to stderr.
```

Now suppose we redirect stdout (that is, filehandle[1]) of outpgm:

```
C:\> outpgm >fileb
```

These are the steps that COMMAND.COM performs:

1. Notices redirection of stdout via '>' on command line.
2. Opens file for writing (or at least writing).
3. Closes stdout.
4. Duplicates the file handle obtained in Step 2 into the slot vacated by stdout (that is, slot 1).
5. Releases (close( )) the filehandle associated with Step 2 because it is not needed anymore.
6. Spawns outpgm, which executes oblivious to the redirection.

Again, notice that outpgm has no knowledge of COMMAND.COM's manipulation of the file handle table. Outpgm just sends output to stdout and stderr, wherever they may be logically connected. When outpgm terminates, COMMAND.COM reconnects stdin, stdout, and stderr back to the console if necessary. It should be noted that this "reconnection" is performed much more elegantly in a Unix environment.

Unfortunately, Microsoft's COMMAND.COM can redirect only stdin and stdout; it cannot, for example, redirect stderr output. But now with the information that I've given you, a program can be written to correct this deficiency. Listing 1, fhdisp.c, shows how to do that. It also adds a graphical depiction of the contents of the first ten elements of the file handle table to demonstrate each step of the redirection:

Usage:

```
FHDISP   [ dos_cmd_line  [error_file] ]
```

where

```
dos_cmd_line
```

is the string that is given to a spawned copy of COMMAND.COM.

```
error_file
```

is the file that stderr will be redirected to if the user desires it. The default is the console.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here is a sample run of fhdisp:

```
C:\> fhdisp
```

0	1	2	3	4	5	6	7	8	9
1	1	1	0	2	.	.	.	.	.

Now let's see what happens when we ask COMMAND.COM to redirect stdin. Let's use the DOS bit bucket, logical file NUL:, as our redirected stdin stream.

```
C:\> fhdisp <nul:
```

0	1	2	3	4	5	6	7	8	9
3	1	1	0	2	.	.	.	.	.

Notice the value in filehandle[0] for stdin. It is no longer 1, but is 3, because of the redirection. Now let's try redirecting stdout:

```
C:\> fhdisp >ftable
C:\>
```

Since the output of fhdisp was redirected to a file (ftable), we'll have to "type it out" to see it:

```
C:\> type ftable
```

0	1	2	3	4	5	6	7	8	9
1	3	1	0	2	.	.	.	.	.

Again, notice the different value for filehandle[1] (stdout). It was redirected by COMMAND.COM to refer to the file "ftable." How about the following example, which redirects both stdin and stdout:

```
C:\> fhdisp <nul: >ftable
C:\> type ftable
```

0	1	2	3	4	5	6	7	8	9
3	4	1	0	2	.	.	.	.	.

Now since both stdin and stdout were redirected by COMMAND.COM, notice that filehandle[0] *and* filehandle[1] have different values (3 and 4).

To further clarify the basic redirection algorithms, Figure 1 demonstrates fhdisp in its "demo mode."

Put That in Your Pipe

Here's an interesting observation for all you MS DOS developers out there in *PJ* land:

```
C:\> type filexx | fhdisp
```

0	1	2	3	4	5	6	7	8	9
3	1	1	0	2	.	.	.	.	.

```
C:\> fhdisp "masm fun.asm;" errors

Before Opening the File:
```

0	1	2	3	4	5	6	7	8	9
1	3	1	0	2	.	.	.	.	.

```
After Opening the File:
```

0	1	2	3	4	5	6	7	8	9
1	3	1	0	2	4	.	.	.	.

```
After Performing the dup2():
```

0	1	2	3	4	5	6	7	8	9
1	3	4	0	2	4	.	.	.	.

```
After Closing the Unwanted File:
```

0	1	2	3	4	5	6	7	8	9
1	3	4	0	2	.	.	.	.	.

```
Hit <cr> to continue:
Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.
All rights reserved.

  50886 Bytes symbol space free

      1 Warning Errors
     11 Severe  Errors
```

FIGURE 1.

LUCID, INC.

Learning UNIX/C In-Depth

"FOR THE CLARITY YOU EXPECT FROM TRAINING"

Ask about our HANDS-ON course for "C" Program Development Under MS/DOS"

For Free Brochure, Information or Registration Please Write or Call: Alice Moss — Director

LUCID, INC.

260 Fifth Avenue
New York, N.Y. 10001
Suite 901

(212) 685-7106

Unix is a trademark of AT & T Bell Labs
MS-DOS is a trademark of Microsoft Corp.

Number 59 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PC/VI

Full Screen Editor for MS-DOS (PC-DOS)

Looking for an Ultra-Powerful Full-Screen editor for your MS-DOS or PC-DOS system? Are you looking for an editor FULLY COMPATIBLE with the UNIX*VI editor. Are you looking for an editor which not only runs on IBM-PC's and compatibles, but ANY MS-DOS system? Are you looking for an editor which provides power and flexibility for both programming and text editing? If you are, then look no further because **PC/VI** IS HERE!

The following is only a hint of the power behind **PC/VI**: English-like syntax in command mode, mnemonic control sequences in visual mode; full undo capability; deletions, changes and cursor positioning on character, word, line, sentence, paragraph or global basis; editing of files larger than available memory; powerful pattern matching capability for searches and substitutions; location marking; joining multiple lines; auto-indentation; word abbreviations and MUCH, MUCH MORE!

The **PC/VI** editor is available for IBM-PC's and generic MS-DOS based systems for only $149. For more information call or write:

Custom Software Systems
P.O. Box 678
Natick, MA 01760
617-653-2555

The UNIX community has been using the VI editor for years. Now you can run an implementation of the same editor under MS-DOS. Don't miss out on the power of **PC/VI**!

*UNIX is a trademark of AT&T Bell Laboratories.

Number 66 on the Reader Service Card

If BASIC's data limit is the question . . .

DataSpace

. . . is the answer

DataSpace gives MS* and GW-BASIC* users . . .

- the ability to define and access string, integer, single, and double precision data arrays
- 10 - one, two or three-dimensional arrays up to 64K each - 640K total
- array storage and retrieval from files in any disc directory
- full compatibility with interpreters or compilers

. . . all with absolutely no impact on BASIC's own data space (requires MS-DOS* 2.10 or higher)

The only answer around for $75 . . .

DESIGN SOLUTIONS INC.

732 Sixth Avenue
Aurora, IL 60505
(312) 898-0651

* MS, GW-BASIC, and MS-DOS are trademarks of Microsoft Corporation

Number 41 on the Reader Service Card

Recalling that the kludgey serialized implementation of MS DOS pipes uses temporary disk files (instead of "parallel"-executing kernel data structures as in Unix)—the first component of the pipeline, the "type" process, writes to stdout. But notice the contents of the filehandle table when the second task starts up. Filehandle[0] no longer refers to the same logical file as filehandles 1 and 2. In fact, stdin of the second process (fhdisp) was redirected by COMMAND.COM to take its input from the temporary file generated by the pipe (%PIPE2.$$$). It is interesting to note that the developers of the MS DOS shell, COMMAND.COM, use their own redirection mechanisms to aid in implementing pipes—interesting, but not unexpected, as most Microsoft developers are familiar with the Unix environment which performs this algorithm.

Now You C It, Now You Don't

With fhdisp in hand, let's tackle the so-called problem with Microsoft's MASM tool. Many programmers complain that MASM does not offer a means of saving the assembler's errors in a file or redirecting them to the printer. The reason for this "oversight" is that the error messages are sent to stderr (which is sensible); however, stderr cannot be redirected by the ineffective COMMAND.COM (which is not sensible). But now this "problem" is moot. Witness:

```
C:\> fhdisp "masm prog.asm;" errfile
          :
          :
... graphical demo of redirection taking place
          :                                    (See Figure 1).

Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.
                                                    etc...

  50886 Bytes symbol space free

      1 Warning Errors
     11 Severe  Errors

C:\>
```

If you want to avoid displaying the Microsoft copyright on the screen (in the case of writing a front end to their tools), just redirect it to the bit bucket.

```
C:\> fhdisp "masm prog.asm;" errfile >nul
```

Just don't forget to hit a <cr> to satisfy the "Hit any key to continue." (that is, the getchar( )) in FHDISP. If you don't, the machine will just idly wait for input. Removing lines 88 and 89 that wait for the <cr> will prevent the program from waiting for that carriage return.

To send your error file to the printer, it's a snap. Just use the logical file "prn" instead of an ordinary file.

```
C:\> fhdisp "masm prog.asm;" prn
```

Industrial Strength Stderr Redirection

Now that we understand the basics of redirection, let's eliminate the file handle table display pizazz of FHDISP and write a more robust, flexible, and safety-conscious tool. ERR.C (Listing 2) performs similar stderr redirection and adds some safety checking. Figure 2 shows a sample run using ERR on our "outpgm" program mentioned above.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
C:\> err outpgm errfile
I'm being sent to stdout.

C:\> type errfile
I'm being sent to stderr.
```

FIGURE 2.

On line 53 of ERR.C, I check for the current value of the switchar character. I did this because Microsoft's system( ) library call does not handle the *undocumented* Unix-style switchar "correctly" (yes, I realize it's undocumented!). Listing 3 GETSW.ASM and listing 4 SETSW.ASM show the source code for getting or setting the switchar from a Microsoft C language program. These two assembler routines do rely on an undocumented feature of DOS (everyone knows that switchar wreaks havoc with the standard DOS backup and restore utilities), so be forewarned. If you never use the Unix-style directory delimiter (/), just leave lines 53, 54, 123 and 124 out.

Now that I've cleared my throat of that pet peeve, let's continue. ERR.C also doublechecks if it is permissible to overwrite an existing file with the requested redirection. If the file exists and if the environment variable "noclobber" is set (to anything) via:

```
C:\> set noclobber=1
```

ERR.C will avoid the overwriting and issue a warning to the user. So the user must explicitly erase the indicated errorfile and try again, or alternately, never set the noclobber variable in the first place. For those who don't know, you can "unset" an environment variable via a null value, as in:

```
C:\> set noclobber=
```

Wouldn't it be nice if COMMAND.COM had this facility so that files would not be overwritten accidentally via redirection? The C Shell of the Berkeley version of Unix has this useful feature. I have always compared COMMAND.COM to EDLIN.COM—it's that bad (the DOS 3.2 shell is better but still not there).

Given that these precautions are observed, we perform the redirection through low level means similar to FHDISP.C. Let's try ERR.C out with some Microsoft system tools and make some interesting observations. The file prog.asm is merely an assembler source file of mine that has errors.

```
C:\> err "masm prog.asm;" errors

Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.  etc.

  50886 Bytes symbol space free

      1 Warning Errors
     11 Severe  Errors

C:\>
```


PL/D Compiler

... a high performance system language

A true system language: PL/D (tm) contains the best features of an assembler in a compiler. You control code location, build structured tables, and define your own macros. Subroutine arguments pass through registers for easiest interface to DOS and ROM BIOS and minimum subroutine call overhead. You access all I/O ports, memory, and machine instructions.

Control your destiny: PL/D includes source that self-compiles at 6000+ lines per minute, minimum. This speeds projects and allows re-targeting to new hardware. Major computer companies code their own critical projects in proprietary system development languages for increased control. You can enjoy the same benefit without their development cost.

Structured: PL/D includes the following multi-line, fully-nestable structures:

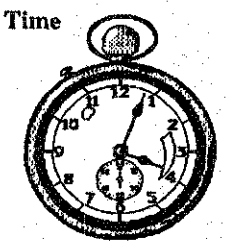
Run-time	Compile-time
IF-THEN-ELSE-ENDIF	IF@-THEN@-ELSE@-ENDIF@
FOR-DO-NEXT	FOR@-DO@-NEXT@
WHILE-THEN-ENDWHILE	WHILE@-THEN@-ENDWH@
CHOOSE-CASE-BUT-ENDCH	CHOOSE@-CASE@-BUT@-ENDCH@
SUBR-ENDSUB	MACRO-ENDMAC (not nested)

$124.95, includes complete source.
81 page user manual, 49 page internals manual. NOT Copy Protected. Add tax in CA.

DAIR Computer Systems VISA MASTER CARD
3440 Kenneth Dr., Palo Alto, CA 94303
(415) 494-7081

Number 5 on the Reader Service Card

Along With Your Computer, Your Time is the Most Important Thing You Own. . .So Why Waste It?



Quilt Programmer Productivity Tools will help you manage your software projects and get control of your time!

SRMS™
Software Revision Management System

SRMS is a full featured revision control system that allows programmers, system managers, and librarians to get control of software projects. SRMS stores every version of a source program in a single ASCII file, with the ability to restore each and every version easily and quickly. Only changes (deltas) between versions are actually stored, so disk usage is drastically reduced. Each version is stored with a unique Version Identification String and unlimited user supplied comments, making identification and documentation tasks easy. SRMS supports projects written in any language, and allows you to use your current editor and compiler without conflict. Multiple programmer projects are offered added protection through the use of "lock" files controlling access to each library. SRMS also incorporates the use of "classes" to bind various versions of different modules together for reconstructing a particular software system release. DOS directory pathnames and environment variable support also aid you in your work. Full typeset documentation with tutorial and project management chapters.
SRMS .....$ 125.00

QMAKE™
Automatic System Generation Utility

QMAKE is a powerful and flexible utility that can be used to control the rebuilding of simple or complex software systems. QMAKE relieves the software developer of the task of remembering which modules of a system need to be rebuilt based on recent changes, how to rebuild them, and in what order to rebuild them. Once you specify the relationships and commands necessary to rebuild your system, and place them in a control file, called a makefile, QMAKE can be used to rebuild your system through the invocation of a single command.
QMAKE will work with most compilers, assemblers, and linkers. It supports full macro definitions, UNIX makefile compatability, recursive invocations, and interpretation of even the most complex dependancies.
QMAKE will also interface with our Software Revision Management System providing you with a complete set of productivity tools to handle any of your project requirements. Full typeset documentation with tutorial.
QMAKE .....$ 99.00

SRMS + QMAKE ..... $ 198.00

QUILT COMPUTING
7048 Stratford Road
Woodbury, MN 55125
(612) 739-4650

MasterCard VISA
Volume Discounts and Dealer Inquiries Welcome

Number 46 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Tools

For Programmers & Non-Programmers

Get 'State of the Art' performance and save valuable time with these high quality utilities!

Opt-Tech Sort™

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can even read the data. Example: 1,000 records of 80 bytes can be read, sorted and a new file written in less than 10 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or called as a subroutine to over 25 different programming languages.

All the sorting, record selection and reformatting facilities you need are included. A partial list of features includes: The ability to process files of any size. Numerous filetypes are supported including Sequential, Random, Delimited, Btrieve, dBASE II & III and many others. Up to 10 key fields can be specified (ascending or descending order). Over 16 different types of data supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes.

MS-DOS $149. ★ NEW ★ Xenix $249.

★ NEW ★ VERSION **On-Line Help™** ★ NEW ★ VERSION

On-Line Help allows you to easily add "Help Windows" to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable Help Systems.

The resident version allows you to add help to any system. Your Help System is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire.

The callable version allows you to easily display help windows from your programs. A simple call to the help system makes the window appear. The original screen is automatically restored when the help window is cleared. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location.

MS-DOS $149. **Demo $10.** (apply toward purchase).

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit your previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted.

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing

P.O. Box 678 — Zephyr Cove, NV 89448

(702) 588-3737

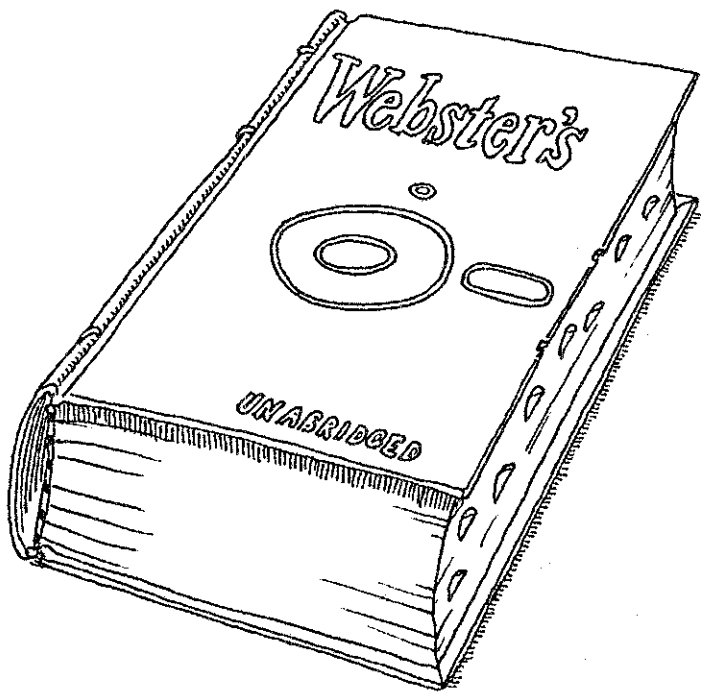
Number 16 on the Reader Service Card

Judging from the above output, the Microsoft MASM developers apparently use stdout for the copyright notice set and stderr for the actual error messages. Let's try Microsoft's Unix-like C compiler driver cl on a program that definitely has some C syntax errors (pgm.c):

```
C:\> err "cl pgm.c" errors

pgm.c
pgm.c(6) : error 146: syntax error : missing ';'

C:\> type errors
Microsoft (R) C Compiler  Version 4.00
Copyright (C) Microsoft Corp 1984, 1985, 1986.  etc..
```

Hmmm . . . So Microsoft's C compiler developers use stderr for the copyright notice, set and stdout for the actual error messages. That's opposite the philosophy of the MASM developers! Now for the linker (the file nope.obj is a file of mine that cannot be linked by itself).

```
C:\> err "link nope;" err1

Microsoft (R) Overlay Linker  Version 3.51
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986.  etc.


Unresolved externals:


_getswitchar in file(s):
NOPE.OBJ(err.c)
_setswitchar in file(s):
NOPE.OBJ(err.c)

C:\> type err1

There were 2 errors detected
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The developers of the Microsoft linker decided to do a mixture of things. The copyright set was sent to stdout, but so was the unresolved externals (error) message. However, further empirical evidence shows that most other error messages were sent to stderr.

Let's hope that Microsoft becomes more consistent in its approach to handling output data streams from its systems software. It would make life easier for developers who are writing front end interfaces to these tools.

EOF—End Of Fun

Whew! Now *that* was a lot of writing for one sitting. I'll let you continue experimenting with redirection. Since I've opted for clarity over efficiency, you can make many obvious enhancements to my code. Some ideas that come to mind include:

1. Allow for full redirection of all 20 data streams with file appending capability.
2. Allow for the override of NOCLOBBER.
3. Avoid the overhead of using system( ).
4. Simplify the code by using the much higher level freopen( ) function (assuming you understand the lower level actions).
5. Try some oddball command lines like:

```
err "err \"masm prog.asm;\" err1"  err2  >nul:
```

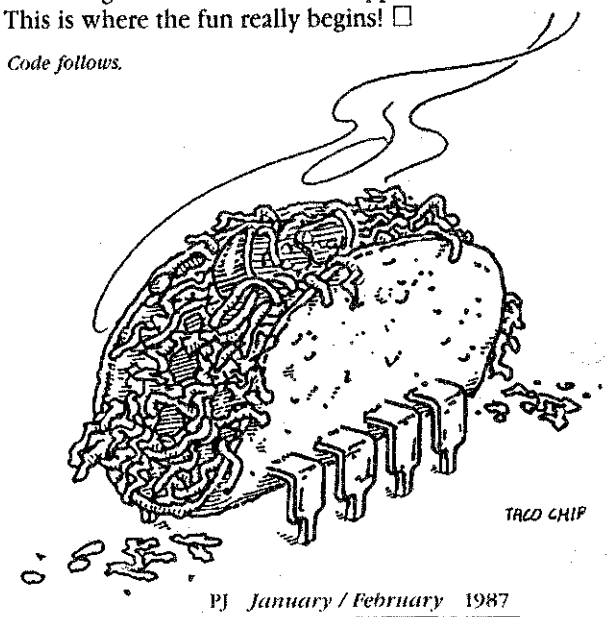
or

```
fhdisp outpgm errors <con: >con:
```

These two command lines are not exactly utilitarian, but it's worthwhile to unravel them as academic experiences in learning DOS and COMMAND.COM internals.

It's a good idea to learn the architecture of simple file redirection; all modern operating systems implement it in some fashion or another. Understanding file redirection will help you in learning another type of redirection, command redirection. Command redirection, *true* parallel piping mechanisms in the Unix tradition, will surely be a feature of the kernel of the multi-tasking version of DOS. That, in addition to some different sorts of inter-task communication schemes such as Microsoft's Dynamic Data Exchange, will be where really interesting and usable concurrent applications will be written. This is where the fun really begins! □

Code follows.



TURBO SOURCE SEARCH (TSS)

TSS is a **RBBS-by-mail** for **TURBO** Pascal. Stop paying AT&T, MCI, SPRINT et al. 10 times what it will cost you if you get it by mail from TSS.

Join **TSS** for $25. Then buy our library disks, **$2 per disk.** (45 disks and growing!!) Archived files = **700+ Kb** of code per **TSS** disk. **TSS** gives more bytes/buck! **Why pay more to get less?**

For your $25 you get
- 2 **FREE** library disks (utilities & catalog)
- 2 **FREE TSS** disks with your first order
- a validated password for the **online TSS-BBS**

TSS-BBS will be an "e-mail" drop-off for members to place orders then log off. It will also serve more traditional RBBS needs. Members will be able to DL the **TSS** disks **(at 10 times the cost of mails!)** if they really MUST have it now!

Join the **TSS BBS-thru-the-mail** for the best in public domain and shareware **TURBO** code.

TURBO SOURCE SEARCH
P.O. Box 876, Scituate, MA 02066, (617) 545-6677
payment: VISA/MC/COD if validated, otherwise check/money orders

Number 35 on the Reader Service Card

$99 BUSINESS BOARD

24 hour business information center using modems at 300/1200/2400 baud. Has a multi-level menu system that is easily customized with no programming req'd. Includes remote PC operations and integrated data base management. Source code additional charge.

$99 PC COMMX $119 CP/M

Emulates VT100, Wyse, HP, ADM, TV, IBM, ADDS, file transfers: KERMIT, XMODEM, COMMX mainframe, Telex/TWX. Instant DOS/foreground switch! Unattended macro controls and 700 entry dial directory. Electronic mail sub-system!

$59 C DATA ENCRYPTION

Data Encryption Standard (U.S. government standard FIPS PUB46) in Microsoft "C". Includes compression & telecomm formatting, allowing for faster transmission & storage on any computer or service. Complete "C" source code provided for additional $249.

HAWKEYE GRAFIX Inc
Box 1400, Oldsmar
Florida 33557
Call 813-786-8161

Number 57 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PORTABLE POWER WITH MasterFORTH

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest.

If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in assembler to make it even faster. MasterFORTH's relocatable utilities and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint and trace your way through most programming problems. A string package, file interface and full screen editor are all standard features. And the optional target compiler lets you optimize your application for virtually any programming environment.

The package exactly matches *Mastering Forth* (Brady, 1984) and meets all provisions of the Forth-83 Standard.

MasterFORTH standard package	$125
(Commodore 64 with graphics)	$100
Extensions	
Floating Point	$60
Graphics (selected systems)	$60
Module relocator (with utility sources)	$60
TAGS (Target Applic. Generation System) – MasterFORTH, target compiler and relocator	$495
Publications & Application Models	
Printed source listings (each)	$35
Forth-83 International Standard	$15
Model Library, Volumes 1-3 (each)	$40

(213) 821-4340

MICROMOTION

8726 S. Sepulveda Bl., #A171
Los Angeles, CA 90045

Number 13 on the Reader Service Card

Greco DOS listing 1

```
/*
 * FHDISP.C        Program to display the File Handle (Descriptor) Table
 *
 * Copyright 1986  Frank D. Greco
 *
 *
 * Compiled with Microsoft 3.0 C Compiler:
 *
 *             cl -Ze fhdisp.c
 *
 * Compiled with Microsoft 4.0 C Compiler
 *
 *             cl fhdisp.c
 *
 *
 * Note that if switchar is set to Unix-style, system() will NOT work!
 */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <process.h>
#include <dos.h>


#define FH_TABLE        0x18            /* PSP offset of filehandle table*/
#define NL              putchar('\n')
#define NUMBER()        for (i =  0; i < 10; i++)\
                            printf("   %d  ", i);\

#define LINE()          putchar('+');\
                        for (i = 0; i < 10; i++)\
                            fputs("-----+", stdout);

#define BEFORE_OPEN()   puts("\nBefore Opening the File:"); draw();
#define AFTER_OPEN()    puts("\nAfter Opening the File:"); draw();
#define AFTER_DUP()     puts("\nAfter Performing the dup2():"); draw();
#define AFTER_CLOSE()   puts("\nAfter Closing the Unwanted File:"); draw();

#define ERR_NOERR       0
#define ERR_BADARGS         1
#define ERR_CANTOPEN    2

extern int _psp;            /* Contains Segment address of the PSP */
                            /* Defined by Microsoft */

char far *filehandle;       /* Needed to access the PSP, MS C Specific */

/*****************************************************************************/

main(argc,argv)
char **argv;
{
/*
 * Use Microsoft Library Macros to setup the far ptr to point to the
 * file handle table.
 */
   FP_SEG(filehandle) = _psp;
   FP_OFF(filehandle) = FH_TABLE;

/*
 * Check sanity of usage...
 */
   if ( argc > 3 )
   {
       fprintf(stderr, "\7Usage: fhdisp [DOS_Command_Line] [file]\n");
       fprintf(stderr, "\nThis will display the filehandle table.\n");
       fprintf(stderr, "If %s requested, redirection demonstrated.\n",
                                "[DOS_Command_Line]" );
       exit(ERR_BADARGS);
   }

   if ( argc == 1 )
   {
       NL;
       draw();                      /* Display process's filehandle table*/
   }

   else if ( argc == 2 )            /* If no third argument, just spawn. */
       system(argv[1]);             /*  Stderr is not redirected  */

   else
   {
       do_redir(argv[2], stderr);   /* Perform redirection */
                                    /* Show filehandle table at each step*/

       printf("Hit <cr> to continue: ");
       getchar();                   /* Wait for a keypress */
/*
 * Next, spawn a child process that writes to stderr.
 * Stderr output will be in requested file obtained from command line.
 */
       system(argv[1]);
   }
   exit(ERR_NOERR);
}
/*****************************************************************************/
/*
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Listing 1 cont.

```
100   * This function will demonstrate what happens at each step of the
101   * redirection process.
102   * This is not the most straightforward or easiest method of redirection.
103   * However once you understand the lower level steps, the easier methods
104   * ( such as with the library call "freopen()" ) will become much clearer.
105   */
106  do_redir(cp, streamfh)
107  char *cp;
108  FILE *streamfh;
109  {
110     int fh;
111
112     BEFORE_OPEN();
113     if ( (fh = open(cp, O_WRONLY|O_TRUNC|O_CREAT,
114                                  S_IREAD|S_IWRITE)) == -1)
115     {
116         fprintf(stderr, "Cannot open file: '%s'\n", cp);
117         exit(ERR_CANTOPEN);
118     }
119     AFTER_OPEN();
120
121     dup2(fh, fileno(streamfh));            /* Do the Redirection */
122     AFTER_DUP();
123
124     close(fh);                          /* Close the unwanted filehandle */
125     AFTER_CLOSE();
126  }
127
128  /*************************************************************************/
129  draw()
130  {
131     int i;                          /* Loop Index */
132     char ch;                        /* Temporary variable for for-loop */
133
134     NUMBER(); NL;                   /* Number the File Handle Table */
135     LINE();   NL;                   /* Draw a Horizontal line */
136
137
138     putchar('|');                   /* Nicely display the file handle table */
139     for (i =  0; i < 10; i++)       /* Restrict to 1st 10 file handle slots */
140     {
141         ch = filehandle[i];
142         if ( ch  >=  0 )
143             printf("  %2d |", ch );
144         else
145             printf("   .  |",  ch );
146     }
147     NL;
148  /*
149   * Note: you could have written the body of the previous for-loop as:
150   *
151   * printf( filehandle[i] >= 0 ? "  %2d |" : "   .  |", filehandle[i] );
152   *
153   */
154     LINE();   NL;                   /* Draw a Horizontal line */
155  }
```

Greco DOS listing 2

```
1   /*
2    * err.c        Program to redirect standard error I/O stream
3    *
4    *             Copyright 1986 Frank D. Greco
5    *
6    *
7    * Compiled by:     Microsoft C Compiler Version 4
8    *
9    *                  C:\> cl err.c getsw setsw;
10   *
11   *
12   * getsw.asm and setsw.asm were
13   *
14   * Assembled by:    Microsoft MASM Version 4.0
15   *
16   *                      C:\> masm getsw;
17   *                      C:\> masm setsw;
18   */
19
20  #include <stdio.h>
21  #include <fcntl.h>
22  #include <sys/types.h>
23  #include <sys/stat.h>
24
25  #define ERR_NOERR       0   /* No Error */
26  #define ERR_NEARGS      1   /* Not Enough Arguments */
27  #define ERR_OVRWRITE    2   /* Attempt to overwrite a file */
28  #define ERR_CANTOPEN    3   /* Can not open the requested error-file */
29
30  main(argc,argv)
31  char **argv;
32  {
33     int fh;              /* Filehandle for the error-file */
```


#1 C interpreter

C-terp

The professional C development environment

Your C compiler creates great final code . . . but as a programming tool, it's too, t o o s l o w.

With C-terp you can edit, debug, and run without the wait. Nothing, but nothing, is faster for developing professional C programs.

Choose the perfect C-terp companion for your C compiler

C-terp/Microsoft	C-terp/XENIX
C-terp/Lattice	C-terp/Aztec
C-terp/Mark Williams	C-terp/C86

Link in all your compiler's functions, your own functions, add-on libraries, assembly routines, and data objects. Get instant access to everything in the C-terp interactive environment.

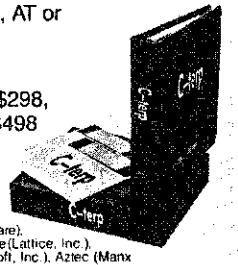
Only C-terp offers all this and more

- **Full K&R with common ANSI enhancements**
- **Source level interactive debugging**
- **Software paging for your big jobs**
- **Complete multi-module support**
- **Run-time pointer checking**
- **Unsurpassed reconfigurable screen editor**
- **Dual display and full graphics support**
- **Large model** ■ **Call-in**

ORDER C-terp TODAY (specify compiler)

C-terp runs on IBM PC, AT or compatibles.

Price:
MS-DOS 2.x and up - $298,
Xenix System V 286 - $498
MC, VISA, COD
30-day money-back
GUARANTEE



Trademarks: C-terp (Gimpel Software), C86 (Computer Innovations), Lattice(Lattice, Inc.), Xenix, Microsoft, MS-DOS (Microsoft, Inc.), Aztec (Manx Software), Mark Williams (Mark Williams Company), IBM (International Business Machines, Inc.)

GIMPEL SOFTWARE

3207 Hogarth Lane, Collegeville, PA 19426

(215) 584-4261

Number 11 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DeSmet C
SUPPORTS
ROMABLE CODE

DeSmet C *$109

Data & Stack in RAM
Code executes out of ROM
Full K&R + V7 extensions
Inline asm
Assembler, linker, librarian
Full screen editor (SEE[tm]),
8087 & S/W floating point

DeSmet C $159
with D88 Debugger

Set Breakpoints by line number or function name
Examine Global/Local variables by name
Show C source while debugging
Both D88 and User displays

Large Memory $209
DeSmet C w/ D88

32-Bit Pointers (Full Megabyte Addressing)
Fast Access of Static Data
Includes the standard compiler features mentioned above.

*Both D88 & Large Memory available as options at $50 each.
We also have a C compiler for the Mac. $150. Call for details.

— plus —

Graphics $35
BASICA-like (+src)

Hacker $25
Source for start-up, RAM.COM & more

Make $50
Full UNIX-level

Tools (+source) **$35**

XARRAY $39
Large Arrays (+source)

C WARE
CORPORATION
P.O. Box C
SUNNYVALE, CA 94087 USA
(408) 720-9696 TELEX: 358185
Street Address: 505 W. Olive, #767
VISA, MC & AMEX accepted

Number 38 on the Reader Service Card

Listing 2 cont.

```
    char oldsw;         /* For switchar detection */

/*
 * Check proper invocation of the command.
 */
   if ( argc == 1 || argc > 3 )
   {
       fprintf(stderr, "\7Not correct number of arguments.\n");
       fprintf(stderr, "Usage: err \"DOS command line\" error-file\n");
       exit(ERR_NEARGS);
   }

/*
 * Check to see if switchar set "properly".  The Microsoft C system()
 * interface routine does not like the Unix-style switchar (i.e., '-').
 * If set to Unix-style, set it to MS-DOS-style and save it
 * for later restoration.
 * These DOS functions are undocumented, so beware!
 */
   if ( (oldsw = getswitchar()) == '-' )
       setswitchar('/');

/*
 * IF user wants stderr in a file
 * THEN
 *         Set the redirection up.
 * ELSE
 *     Default is CON:
 * ENDIF
 */
   if ( argc == 3 )
   {

/*
 * Now check to see the following:
 *
 * IF  error-file already exists  AND  env. variable 'NOCLOBBER' is SET
 * THEN
 *     Warn user that redirection will destroy previous file
 *     Exit back to DOS
 * ENDIF
 */

       if ( !access(argv[2], 0) )      /* If 'error-file' already exists*/
           if ( getenv("NOCLOBBER") )  /* If 'noclobber' set, exit out */
           {
               fprintf(stderr, "%s \n%s '%s'.\n",
                               "*Warning*  Noclobber is set.",
               "Redirection of stderr will OVERWRITE file: ",
                               argv[2]);
               exit(ERR_OVRWRITE);
           }
/*
 * Request DOS to open the specified error file.
 */
       fh = open(argv[2],  O_WRONLY|O_TRUNC|O_CREAT,  S_IREAD|S_IWRITE);
       if ( fh == -1 )
       {
           fprintf(stderr, "Cannot open file '%s'\n", argv[2]);
           exit(ERR_CANTOPEN);
       }

/*
 * The  dup2()  function is the heart of this program.
 * As called, it will duplicate the filehandle received
 * from the previous open(), that is, 'fh',  and insert
 * it in the filehandle slot reserved for stderr, that is,
 * filehandle #2.  Note that now filehandle #2 no longer
 * refers to the screen, it refers to the opened errorfile.
 */
       dup2(fh,2);

/*
 * Since we do not need filehandle #fh anymore, it is closed.
 * This also prevents an unused but allocated filehandle
 * to be passed to a child process who would have inherited it.
 */
       close(fh);
   }
/*
 * At this point you could continue in this manner and perform
 * possible stdin and stdout redirection.
 * Since you should get the idea by now, we'll use COMMAND.COM to
 * assist us in performing any further stdin/stdout redirection.
 * This is not efficient, but it's effective.
 */

   system(argv[1]);            /* Spawn a DOS shell process */

   if (oldsw == '-')           /* If switchar was reset, */
       setswitchar(oldsw);     /* Then reset it back to original char */

   exit(ERR_NOERR);            /* Good Housekeeping Practice */
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Greco DOS listing 3

```
;
; setswitchar(switchar) - Sets the switchar to desired character.
; char switchar;          To be linked with Microsoft C Compiler
;                                          Version 4.0
;
; Assembled by Microsoft MASM Version 4.0:
;
;       C:\> masm setsw;
;

_TEXT   segment public byte 'code'
        assume cs:_TEXT

_setswitchar    proc    near
        public  _setswitchar

        push    bp
        mov     bp, sp

        mov     dl, 4[bp]       ; Get switchar argument from stack
        mov     ax, 03701h      ; Function 0x37 Get/Set Switchar
                                ; AH: 1     Set Switchar
        int     021h            ; Call DOS
        mov     ax, 4[bp]       ; Make return value = to switchar

        pop     bp
        ret

_setswitchar endp
_TEXT   ends
        end
```

Greco DOS listing 4

```
;
; getswitchar() -  Gets the current switchar.  To be linked with
;                  Microsoft C Compiler Version 4.0
;
; Assembled by Microsoft MASM Version 4.0:
;
;       C:\> masm getsw;
;

_TEXT   segment public byte 'code'
        assume cs: _TEXT

_getswitchar:
        public  _getswitchar

        mov     ax, 03700h      ; Function 0x37 Get/Set Switchar
                                ; AH: 0     Get Switchar
        int     021h            ; Call DOS

        mov     al, dl          ; Move Switchar to AX for function
                                ; return value convention.
        ret
_TEXT   ends

        end
```


NEW **FILEMOVER**

by California Jack Cassidy

$59.95 (IBM PC, XT, AT or clone, 256K min.)

Not copy-protected, includes Source Code

FILEMOVER™ is perfect for Copying or Moving all types of files from disk to disk (or directory to directory). Reorganize your disk library and make back-ups without hassle. FileMover is easy to use and menu driven. Hard disk compatible too. **Subdirectory structure is preserved** when files are copied. And, if your destination disk fills up, you can continue on another disk.

FILE SORTER: Automatically alphabetize directories by file name and/or extension. Transfer files to other disks in any order.

FILE DISGUISER: Hide/unhide files so only you know they exist. Alter file names, dates and times. Convert files to Read-Only status. Inspect files in Hex-ASCII format...

QUICKSORT: DOS's sorter is extremely s-l-o-w on anything but small jobs. FileMover's sorter is one of the *fastest* anywhere!

Bonus–FOREVER FILE DELETER

Did you know that files you have "deleted" often *stay on your disks*, just waiting for someone to snoop through or undelete? FileMover's Super-Deleter totally ZEROES OUT unwanted files so they don't exist. Period.

Free Turbo Command Chart

All of those *Turbo Pascal*™ functions and procedures on one chart that always stays in front of you. A great PC programming tool.

NEW **MINIPIX DISK#2**

200 New Graphics for IBM PRINT SHOP™. . . $34.95

Beagle Bros MICRO SOFTWARE

3990 Old Town Ave. / San Diego, CA 92110

619-296-6400

Products available at Software Stores or by mail:

Order Toll Free (Mon.-Fri., 9am-4pm Pacific Coast time)

1-800-345-1750 (Calif: 1-800-992-4022)

Add $2.50 shipping ($5.00 overseas), $3 COD, 6% if Calif.

All items in stock and shipped immediately.

Number 84 on the Reader Service Card

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Inside the EGA

by
Michael Abrash

This is the first of a series of articles about graphics. Actually, it's about more than just graphics—it's about algorithms and assembly-language programming and all manner of tricks for putting IBM PC-family microcomputers through their paces. Still, the focus will be on graphics of the EGA/CGA/Hercules sort, all the while keeping an eye on higher-performance boards that have the potential to become standards someday soon. If there's any particular area you'd like to see covered or interesting findings you'd like to share, let me know.

I'm going to start with a look at the Enhanced Graphics Adapter (EGA). The CGA and Hercules boards are pretty simple, and a great deal has been written about how to program them. The EGA, on the other hand, is a difficult board to program efficiently (the more so because IBM's documentation is hard to obtain and harder to understand), and little information about programming the EGA has seen print. The EGA's tremendous potential—far greater than that of the CGA—makes it well worth knowing about, and that's what the next few articles will be about.

The EGA

Although the EGA is fast becoming the standard graphics adapter in the IBM PC marketplace, it has not met with universal approval. The knock against the EGA is that its performance is inadequate. True, the EGA is not an ultra-high-resolution board, but it offers a solid price/performance ratio. Sixteen colors at a resolution of 640 x 350, complete with a monitor and decent compatibility with older software, for under $800 is a pretty good deal—that's about what I paid for 320 x 200 in 4 colors just four years ago. (I'm talking about EGA clones, of course—the IBM EGA and monitor are greatly overpriced.) In my opinion, the EGA provides adequate resolution and color for the majority of PC applications at a price most users can afford.

There's another facet of graphics performance, though, and that's the speed at which programs can update the screen. Bit-mapped graphics tend to be slow when handled by the main processor: Witness Microsoft Windows, which really requires an AT for decent speed, or, for that matter, the Macintosh, which expends much of the power of an 8-MHz 68000 in supporting a bit-mapped interface. The time spent in bit-map manipulation goes up rapidly as the size of the bit map and the complexity of controlling it increase. The EGA's bit map is 16 times as large as the CGA's, and although the EGA has many useful features built in, those features are controlled through dozens of registers. The result: It's hard to produce fast graphics on the EGA.

Hard, but not impossible—and that's why I like this odd board. It's a throwback to an earlier generation of micros, when inventive coding and a solid understanding of the hardware were the best tools for improving performance. Increasingly, faster processors and powerful coprocessors are seen as the solution to the sluggish software produced by high-level languages and layers of interfacing and driver code, and that's surely a valid approach. However, there are hundreds of thousands of EGAs installed right now, with no coprocessors to help and hardly an 80386 in sight. What's more, because the EGA is an 8-bit device, and because of display memory wait states, an AT isn't as much a help as you'd expect. The upshot is that only a good low-level coder who understands the EGA can push the board to its potential.

I'll be exploring the EGA by selecting a specific algorithm or feature and implementing code to support it on the EGA, examining aspects of the EGA architecture as they become relevant. You'll get to see EGA features in context, where they are more comprehensible than in IBM's somewhat arcane documentation, and you'll get working code to use or to modify to meet your needs. If there's a particular aspect of the EGA that you'd like to read about, send me a letter and I'll try to work it in.

The prime directive of EGA programming is that there's rarely just one way to program the EGA for a given purpose. Once you understand the tools the EGA provides, you'll be able to combine them to generate the particular synergy your application needs. My EGA routines are not intended to be taken as gospel, or to show "best" implementations, but rather to start you down the road to understanding the EGA. Let's begin.

An Introduction to the EGA

Most discussions of the EGA start out with a traditional "Here's a block diagram of the EGA" approach, with lists of registers and statistics. I'll get to that eventually, but you can find it in IBM's EGA documentation and several other magazines. Besides, it's numbing to read specifications and explanations, and the EGA is an exciting board. It's the sort of board that makes you want to get your hands dirty under the hood, to write some nifty code just to see what the board can do. What's more, the best way to understand the EGA is to see it work, so let's jump right into a sample of the EGA in action, getting a feel for the EGA's architecture in the process.

Listing 1 is a sample EGA program that pans around an animated 16-color high-resolution (640 x 350) playfield. There's a lot packed into this code; I'm going to take advantage of the technically high level of *PJ* readership and assume you can figure out the non-EGA aspects for yourself. I'm not going to explain how the ball is animated, for example. (For an introduction to animation, see "Animation Techniques for the IBM PC," by Michael Abrash and Dan Illowsky, *PC Tech Journal*, July 1986.) What I will do is cover each of the EGA features used in this program—virtual screen, vertical and horizontal panning, color plane manipulation, multi-plane

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

block copying, and page flipping—at a conceptual level, letting the code itself demonstrate the implementation details.

Some Background

A little background is necessary before we can begin to examine Listing 1. The EGA is built around four VLSI (Very Large Scale Integrated) chips, named the CRT Controller (CRTC), the Timing Sequencer (TS), the Attribute Controller (ATC), and the Graphics Data Controller (GDC). There are two GDCs per EGA since each GDC controls two of the EGA's four planes of memory. Some EGA compatible chip sets combine several of these chips into a single chip or shuffle specific functions among chips, but the programming interface always looks the same since otherwise the chip set would not be EGA-compatible.

Each of these chips has a sizeable complement of registers. It is not particularly important that you understand why a given chip has a given register; all the registers together make up the programming interface, and it is the entire interface that is of interest to the EGA programmer. However, the means by which most EGA registers are addressed makes it necessary for you to remember which registers are in which chips.

Most EGA registers are addressed as internally indexed registers. The internal address of the register is written to a given chip's index register, and then the data for that register is written to the chip's data register. For example, GDC register 8, the Bit Mask register, is set to 0FFh by writing 8 to port 3CEh, the GDC Index register, and then writing 0FFh to port 3CFh, the GDC Data register. Internal indexing makes it possible to address the 9 GDC registers through only two ports and allows the entire EGA programming interface to be squeezed into less than a dozen ports. The downside is that two I/O operations are required to access most EGA registers.

The ports used to control the EGA are shown in the table below. The CRTC, TS, and GDC Index registers are located at the addresses of their respective Data registers plus one. However, the ATC Index and Data registers are located at the same address, 3C0h. The function of this port toggles on every OUT to 3C0h and resets on every read from the Input Status 1 register (3DAh when the EGA is in color modes, 3BAh in monochrome modes). Note that all CRTC registers are addressed at either 3DXh or 3BXh, the former in color modes and the latter in monochrome modes. This provides compatibility with the register addressing of the Color/Graphics and Monochrome Adapters.

The ports through which the EGA is controlled

These are the official addresses, but some EGA ports are incompletely decoded and can be addressed at several ports. Most importantly, the ATC Index/Data register can be addressed at 3C1h as well as at 3C0h.

Register	Address
ATC Index/Data register	3C0h (write only)
Input Status 0 register	3C2h (read only)
TS Index register	3C4h (write only)
TS Data register	3C5h (write only)
Graphics 2 Position register	3CAh (write only)
Graphics 1 Position register	3CCh (write only)
GDC Index register	3CEh (write only)
GDC Data register	3CFh (write only)
CRTC Index register	3B4h/3D4h (write only)
CRTC Data register	3B5h/3D5h (write only)
Input Status 1 register/ ATC Index/Data reset	3BAh/3DAh (read only)
Feature Control	3BAh/3DAh (write only)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The method used in the EGA BIOS to set registers is to point DX to the desired index register, load AL with the index, perform a byte OUT, increment DX to point to the data register (except in the case of the ATC, where DX remains the same), load AL with the desired data, and perform a byte OUT. A handy shortcut is to point DX to the desired index register, load AL with the index, load AH with the data, and perform a word OUT. Since the high byte of the OUT value goes to port DX+1, this is equivalent to the first method but is considerably faster. This works for even the ATC since the ATC Index and Data registers are decoded at both 3C0h and 3C1h; however, be sure that the ATC Index/Data register is set to index mode before programming the ATC with a word OUT (you can ensure this by first reading the Input Status 1 register to reset ATC addressing).

How safe is this method of addressing EGA registers? I have run into accelerator boards that had trouble with word OUTs; however, all such problems I am aware of have been fixed. Moreover, Microsoft Windows uses word OUTs, even to the ATC, so any clone computer or EGA that doesn't support word OUTs could scarcely be considered a clone at all.

A speed tip: The setting of each chip's index register remains the same until it is reprogrammed. This means that in cases where you are setting the same internal register repeatedly, you can set the index register to point to that internal register once, then write to the data register multiple times. For example, the Bit Mask register (GDC register 8) is set repeatedly inside a loop when drawing lines. The standard code for this is

```
MOV     DX,03CEH   ;point to GDC Index register
MOV     AL,8       ;internal index of Bit Mask register
OUT     DX,AX      ;AH contains Bit Mask register setting
```

Alternatively, the GDC Index register could initially be set to point to the Bit Mask register with

```
MOV     DX,03CEH   ;point to GDC Index register
MOV     AL,8       ;internal index of Bit Mask register
OUT     DX,AL      ;set GDC Index register
INC     DX         ;point to GDC Data register
```

and then the Bit Mask register could be set repeatedly with the byte-size OUT instruction:

```
OUT     DX,AL      ;AL contains Bit Mask register setting
```

which is four cycles faster than a word-sized OUT, and which does not require AH to be set, freeing up a register. Be aware that this method works only if the GDC Index register remains unchanged throughout the loop.

Linear Planes and True EGA Modes

The EGA's memory is organized as four 64K planes. (The base model from IBM comes with only 16K per plane, but this configuration doesn't support the 16-color high-resolution mode. So far as I can tell, this version of the EGA is about as widely used as the PC's cassette port, so I will discuss only 256K EGA operation.) Each of these planes is a linear bit map; that is, each byte from a given plane controls eight adjacent pixels on the screen, the next byte controls the next eight pixels, and so on to the end of the scan line. The next byte then controls the first eight pixels of the next scan line, and so on to the end of the screen. This will no doubt come as a relief to programmers weary of compensating for the CGA's two-bank architecture and the Hercules Graphics Card's four-bank architecture.

The EGA adds a powerful twist to linear addressing; the logical width of the screen in EGA memory need not be the same as the physical width of the display. The programmer is free to define all or part of the EGA's large memory map as a logical screen of up to 4096 pixels in width and then use the physical screen as a window onto any part of the logical screen. What's more, a virtual screen can have any logical height up to the capacity of EGA memory. Such a virtual screen could be used to store a spreadsheet or a CAD/CAM drawing, for instance. As we will see shortly, the EGA provides excellent hardware for moving around the virtual screen; taken together, the virtual screen and the EGA's smooth panning capabilities can generate very impressive effects.

All four linear planes are addressed in the same 64K memory space starting at A000:0000. Consequently, there are four bytes at any given address in EGA memory. The EGA provides special hardware to assist the CPU in manipulating all four planes, in parallel, with a single memory access so that the programmer doesn't have to spend a great deal of time switching between planes. Astute use of this EGA hardware allows EGA software to equal the performance of CGA software even though the EGA bit map is much larger than the CGA bit map.

Each memory plane provides one bit of data for each pixel. The bits for a given pixel from each of the four planes are combined into a nibble that serves as an address into the EGA's palette RAM, which maps the one of sixteen colors selected by display memory into any one of sixty-four colors, as shown in Figure 1. All sixty-four mappings for all sixteen colors are independently programmable.

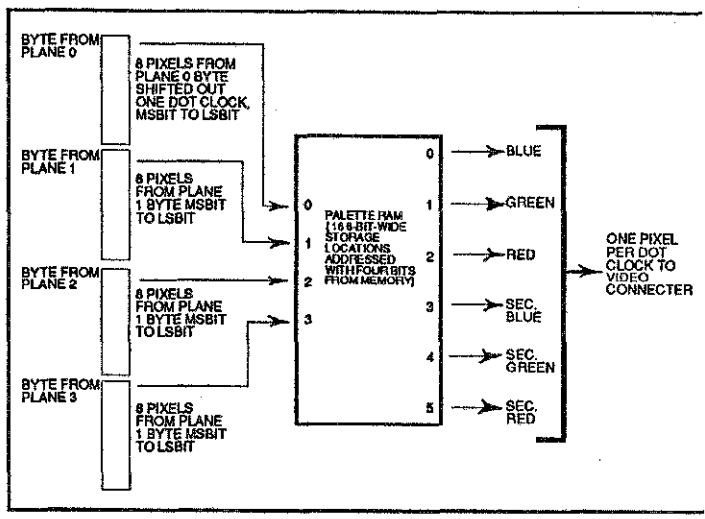

BYTE FROM PLANE 0
8 PIXELS FROM PLANE 0 BYTE SHIFTED OUT ONE DOT CLOCK, MSBIT TO LSBIT
BYTE FROM PLANE 1
8 PIXELS FROM PLANE 1 BYTE MSBIT TO LSBIT
BYTE FROM PLANE 2
8 PIXELS FROM PLANE 2 BYTE MSBIT TO LSBIT
BYTE FROM PLANE 3
8 PIXELS FROM PLANE 3 BYTE MSBIT TO LSBIT
PALETTE RAM (16 6-BIT-WIDE STORAGE LOCATIONS ADDRESSED WITH FOUR BITS FROM MEMORY)
0
1
2
3
4
5
BLUE
GREEN
RED
SEC. BLUE
SEC. GREEN
SEC. RED
ONE PIXEL PER DOT CLOCK TO VIDEO CONNECTER

Figure 1. Video data from EGA display memory to video connector

The EGA BIOS supports several graphics modes (modes 4, 5, and 6) in which EGA memory appears not to be organized as four linear planes. These modes exist for CGA compatibility only and are not true EGA graphics modes. Use them when you need CGA-type operation and ignore them the rest of the time. The EGA's special features are most powerful in true EGA modes, and it is on these modes (modes 0Dh, 0Eh, 0Fh, and 10h) that I will concentrate. EGA text modes, which feature soft fonts, are another matter entirely, to be explored separately at a later date.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

With that background out of the way, we can get on to the sample EGA program shown in Listing 1. I suggest you run the program before continuing since the explanations will mean far more to you if you've seen the features in action.

Smooth Panning

The first thing you'll notice upon running the sample program is the remarkable smoothness with which the display pans from side-to-side and up-and-down. That the display can pan at all is made possible by two EGA features: the 256K bit map and the virtual screen capability. Even the most memory-hungry of the EGA modes, mode 10h, uses only 28K per plane, for a total of 112K out of the total 256K of EGA memory. Consequently, there is room in EGA memory to store more than two full screens of video data. In the sample program, memory is organized as two virtual screens, each with a resolution of 672 x 384, as shown in Figure 2.

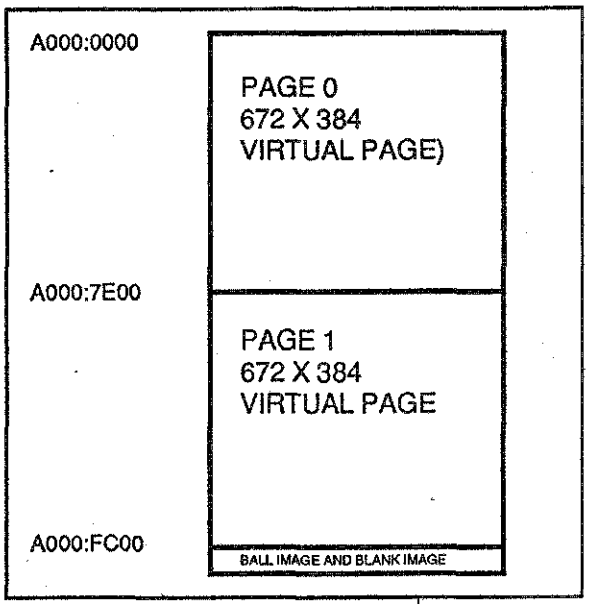


Figure 2. Organization of EGA memory in sample program. Space is reserved for two 672 x 384 virtual pages and for images of the ball and the blank.

The area of the virtual screen actually displayed at any given time is selected by setting the display memory address at which to begin fetching video data; this is set by way of the start address registers (Start Address High, CRTC register 0Ch, and Start Address Low, CRTC register 0Dh). Together these registers make up a 16-bit display memory address at which the CRTC begins fetching data at the beginning of each video frame. Increasing the start address causes higher-memory areas of the virtual screen to be displayed. For example, the Start Address High register could be set to 80h and the Start Address Low register could be set to 00h in order to cause the display screen to reflect memory starting at offset 8000h in each plane, rather than at the default offset of 0.

The logical height of the virtual screen is defined by the amount of EGA memory available. As the EGA scans display memory for video data, it progresses from the start address toward higher memory one scan line at a time until the frame is completed. Consequently, if the start address is increased, lines farther towards the bottom of the virtual screen are displayed; in effect, the virtual screen appears to scroll up on the physical screen.

The logical width of the virtual screen is defined by the Offset register (CRTC register 13h), which allows redefinition of the number of words of display memory considered to make up one scan line. Normally, 40 words of display memory constitute a scan line; after the CRTC scans these 40 words for 640 pixels' worth of data, it advances 40 words from the start of that scan line to find the start of the next scan line in memory. This means that displayed scan lines are contiguous in memory. However, the Offset register can be set so that scan lines are logically wider (or narrower, for that matter) than their displayed width. The sample program sets the Offset register to 2Ah, making the logical width of the virtual screen 42 words, or 42 * 2 * 8 = 672 pixels, as contrasted with the actual width of the hi-res screen, 40 words or 640 pixels. The logical height of the virtual screen in the sample program is 384; this is accomplished simply by reserving 84 * 384 contiguous bytes of EGA memory for the virtual screen, where 84 is the virtual screen width in bytes and 384 is the virtual screen height in scan lines.

The start address is the key to panning around the virtual screen. The start address registers select the row of the virtual screen that maps to the top of the display; panning down a scan line requires only that the start address be increased by the logical scan line width in bytes, which is equal to the Offset register times two. The start address registers select the column that maps to the left edge of the display as well, allowing horizontal panning, although in this case only relatively coarse byte-sized adjustments—panning by eight pixels at a time—are supported. Smooth horizontal panning is provided by the Horizontal Pel Panning register, ATC register 13h, working in conjunction with the start address. Up to 7 pixels' worth of single pixel panning of the displayed image to the left is performed by increasing the Horizontal Pel Panning register from 0 to 7. This exhausts the range of motion possible via the Horizontal Pel Panning register; the next pixel's worth of smooth panning is accomplished by incrementing the start address by one and resetting the Horizontal Pel Panning register to 0. Smooth horizontal panning should be viewed as a series of fine adjustments in the 8-pixel range between coarse byte-sized adjustments.

A horizontal panning oddity: Alone among EGA modes, monochrome EGA text mode has 9 dots per character clock. Smooth panning in this mode requires cycling the Horizontal Pel Panning register through the values 8, 0, 1, 2, 3, 4, 5, 6, and 7. 8 is the "no panning" setting.

There is one annoying quirk about programming the ATC. When the ATC Index register is set, only the lower five bits are used as the internal index. The next most significant bit, bit 5, controls the source of the video data sent to the monitor by the EGA. When bit 5 is set to 1, the output of the palette RAM, derived from display memory, controls the displayed pixels; this is normal operation. When bit 5 is 0, video data does not come from the palette RAM, and the screen becomes a solid color. The only time bit 5 of the ATC Index register should be 0 is during the setting of a palette RAM register since the CPU is able to write to palette RAM only when bit 5 is low. Immediately after the palette RAM is set, however, 20h should be written to the ATC Index register to restore normal video, and at all other times bit 5 should be set to 1.

By the way, palette RAM can be set via the BIOS video interrupt (interrupt 10h), function 10h. Whenever an EGA func-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tion can be performed reasonably well through a BIOS function, as it can in the case of setting palette RAM, it should be, both because there is no point in reinventing the wheel and because the BIOS may well mask incompatibilities between the IBM EGA and EGA clones. For instance, it is not easy to set all the palette RAM registers without causing momentary flicker, and the exact parameters for doing this vary somewhat among EGA clones. However, it's a safe bet that the palette RAM-setting routines in every BIOS are flicker-free.

Color Plane Manipulation

The EGA provides a powerful array of hardware assistance for manipulating the four display memory planes. Two features illustrated by the sample program are the ability to control which planes are written to by a CPU write and the ability to copy four bytes—one from each plane—with a single CPU read and a single CPU write.

The Map Mask register (TS register 2) selects which planes are written to by CPU writes. If bit 0 of the Map Mask register is 1, each byte written by the CPU will be written to EGA memory plane 0, the plane that provides the video data for the least significant bit of the palette RAM address. If bit 0 of the Map Mask register is 0, CPU writes will not affect plane 0. Bits 1, 2, and 3 of the Map Mask register similarly control CPU access to planes 1, 2, and 3, respectively. Any of the sixteen possible combinations of enabled and disabled planes can be selected. Beware, however, of writing to an area of memory that is not zeroed. Planes that are disabled by the Map Mask register are not altered by CPU writes, so old and new images can mix on the screen, producing unwanted color effects as, say, three planes from the old image mix with one plane from the new image. The sample program solves this by ensuring that the memory written to is zeroed. A better way to clear memory is provided by the set/reset capabilities of the EGA, which I'll cover another time.

The sample program writes the image of the colored ball to EGA memory by enabling one plane at a time and writing the image of the ball for that plane. Each image is written to the same EGA addresses; only the destination plane, selected by the Map Mask register, is different. You might think of the ball's image as consisting of four colored overlays, which together make up a multicolored image. The sample program writes a blank image to EGA memory by enabling all planes and writing a block of zero bytes; the zero bytes are written to all four EGA planes simultaneously.

The images are written to a nondisplayed portion of EGA memory in order to take advantage of a useful EGA hardware feature, the ability to copy all four planes at once. As shown above, four times as many reads and writes—and several OUTs as well—are required to copy a multicolored image into EGA memory as would be needed to draw the same image in the CGA's high-resolution mode. This causes unacceptably slow performance, all the more so because the wait states that occur on accesses to EGA memory make it very desirable to minimize display memory accesses.

The solution is to take advantage of the EGA's write mode 1, which is selected via bits 0 and 1 of the GDC Mode register (GDC register 5). (Be careful to set bits 2-7 properly for the current display mode when setting bits 0 and 1.) In write mode 1, a single CPU read loads the addressed byte from all four planes into the EGA's four internal latches, and a single CPU write writes the contents of the latches to the four planes. During the write, the byte written by the CPU is irrelevant.

The sample program uses write mode 1 to copy the images that were previously drawn to the high end of EGA memory into a desired area of display memory, all in a single block copy operation. This is an excellent way to keep the number of reads, writes, and OUTs required to manipulate the EGA's display memory low enough to allow real-time drawing.

The Map Mask register can still mask out planes in write mode 1. All four planes are copied in the sample program because the Map Mask register is still 0Fh from when the blank image was created.

The animated images appear to move somewhat jerkily because they are byte-aligned and so must move a minimum of 8 pixels horizontally. This is easily solved by storing rotated versions of all images in EGA memory and then in each instance drawing the correct rotation for the pixel alignment at which the image is to be drawn.

Page Flipping

CGA graphics typically flicker and/or ripple, an unavoidable result of modifying display memory at the same time that it is being scanned for video data. The extra display memory of the EGA makes it possible to perform page flipping, which eliminates such problems. The basic premise of page flipping is that one area of display memory is displayed while another is being modified. The modifications never affect an area of memory as it is providing video data, so no undesirable side effects occur. Once the modification is complete, the modified buffer is selected for display, causing the screen to change in a single frame. The other buffer is then available for modification.

Graphics mode page flipping is not possible with the CGA because the CGA's display memory is barely large enough to hold a single screen's bit map (a single page). However, the EGA has 64K per plane, enough to hold two pages and more even in hi-res mode. (A hi-res page is 80 times 350, or 28,000 bytes.) For page flipping, two non-overlapping areas of display memory are needed. The sample program uses two 672 x 384 virtual pages, each 32,256 bytes long, one starting at A000:0000 and the other starting at A000:7E00. Flipping between the pages is as simple as setting the start address registers to point to one display area or the other.

The timing of the switch between pages is critical to achieving flicker-free animation. It is critical that the program never be modifying an area of display memory as that memory is providing video data. This means that the start address should be changed at some time during the frame after which the page flip is to occur since the start address for a given frame is latched from the start address registers at the end of the previous frame. The most readily available vertical display status on the EGA is the vertical sync pulse status, available at bit 3 of the Input Status 1 register (addressed at 3BAh/3DAh). A logical approach to page flipping would seem to be to wait for the leading edge of the vertical sync pulse, then set the start address registers.

Unfortunately, the vertical sync bit is not the ideal status to monitor for the end of the frame since it doesn't begin until part way through the vertical retrace period, after the start address has been latched into the linear address counting circuitry. If the sample program were to draw to the supposedly undisplayed page during the frame after the start address is changed to the other page, flicker would occur; at this time, the new start address has not yet taken effect, so that page is actually still being displayed. To avoid this, the sample program

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

waits twice for the leading edge of the vertical sync pulse, once to be sure about when the starting address registers are being set and once to allow the page flip to become effective.

Waiting for the sync pulse has the side effect of causing program execution to synchronize to the EGA's frame rate of 60 per second. In a program where all drawing can be done during a single frame time, this synchronization has the useful consequence of causing the program to execute at the same speed on an AT as on a PC.

As mentioned above, the vertical sync bit is not a perfect status to monitor for the end of a frame. There is a way to detect the true beginning of the EGA's vertical retrace period, via the vertical interrupt; for a sample implementation, see "Software Sprites for the IBM EGA and CGA" by Michael Abrash and Dan Illowsky, *PC Tech Journal*, August 1986.

An important point illustrated by the sample program is that while the EGA's bit map is far larger and more versatile than is the case with other adapters, it is nonetheless a limited resource and must be used judiciously. The sample program uses EGA memory to store two 672 x 384 virtual pages, leaving only 1024 bytes free to store images. In this case, the only images needed are a colored ball and a blank block with which to erase it, so there is no problem, but many applications require dozens or hundreds of images. The tradeoffs between virtual page size, page flipping, and image storage must always be kept in mind when designing programs for the EGA.

Just an Introduction

That pretty well covers the important points of the sample EGA program in Listing 1. To see the program run in 640 x 200 16-color mode, comment out the HIRES_VIDEO_MODE equate; in this mode, the display screen is much smaller relative to the virtual screen, so the smooth panning is far more pronounced. There are many EGA features we haven't even touched on, but the object is to give you a feel for the variety of features available on the EGA, to convey the flexibility and complexity of the EGA's resources, and to point out how different from the CGA the EGA is. Starting with the next article, I'll begin to systematically explore the EGA on a more detailed basis.

EGA Clones

There are an amazing number of EGA clones on the market now. Most are built around the Chips and Technologies chip set, but several other chip sets are in use as well. The EGA clones sell for so much less than the IBM version that it's hard to imagine that anyone but the most conservative corporate purchaser would buy the real thing, particularly since all the clones seem to be more than adequately compatible with the IBM EGA.

The clones are not exactly like the IBM EGA, however. Some have bugs in the chip set implementation, others have slightly different board layouts, and all have BIOS ROMs that differ to some extent from IBM's. Some manufacturers have also made a conscious decision to deviate slightly from the IBM standard in order to build a better board; for example, several manufacturers have provided modes with higher resolution than the EGA can muster.

What this means to the programmer is that EGA code should be tested on several EGA clones before being released. Given the tremendous popularity of the Chips and Technology EGA-clone chip set, it is not enough to design for the IBM EGA; I suspect that there are more C & T-based EGAs out there than IBM EGAs. In truth, the EGA standard is less standard than, say, the PC standard, because the PC is built out of off-the-shelf hardware, not custom VLSI chips, and because the EGA video BIOS is twice as large as the entire PC BIOS. There's no solution to this, but broad testing is a reasonable substitute.

REAL - TIME GRAPHICS ANIMATION

For the IBM PC, XT, AT or compatible, with Color Graphics Adapter and color monitor.

With this plug-in accessory, your PC can produce graphics as fast as expensive dedicated graphics work stations. The card acts as a graphics coprocessor to the CPU. Graphics primitives are drawn by an HD63484 chip. No need to compute the adresses of individual pixels in a line, the card takes the ends of the line as input then draws the line at high speed. The line appears on the screen at a million color pixels per second.

Keep your color graphics adapter and standard RGBI monitor, this card uses both. Since your old adapter remains a part of the system, you are sure that your machine remains completely IBM compatible. Gives you all 16 colors on the screen at the same time, even with 640x200 dots.

Are you tired of having to program graphics applications in assembler and C? This card comes with an include file for Turbo Pascal. Nearly 70 procedures and functions allow you to use a high level language to write fast graphics programs. In addition to the usual lines, circles, ellipses, rectangles, 2-color area fills, etc., this package includes support for real-time animation. You can swap in a hidden screen during vertical retrace. Your animation will be clean and synchronized - no hash the easy way. Automatic checkerboard dithering is supported, giving 256 psuelo colors on a standard IBM color monitor.

Envious of your friends with Amigas and STs? This card can redraw a 21x21 3-D net graph in 1/60 sec. That is 50400 vectors per second. And the card includes support for 3-D figure drawing and rotation.

It even works with your speed up cards and fast mother-boards. No memory addresses are used, and only four bytes of I/O ports are needed, so the board won't conflict with your other graphics adapters or peripherals.

This card is available only directly from the factory. Two cards are available, one with a resolution of 640x200, the other one goes to 640x400 but requires a long persistence phosphor monitor. Both cards come with a 30 day money-back trial period. Try the card, if it doesn't perform as claimed, or if you are unsatisfied with it for any reason send it back for a full refund. The cards also come with a 180 day warranty, and include free software updates.

EHGCp2	640x200	only	$374.00	Check or Money Order
EHGCp4	640x400	interlace	578.00	30 day trial 180 day warranty (714) 261-1725
	Shipping and Handling		4.00	

E-Heart Engineering / 18103 Sky Park South #D / Irvine, CA 92714

Number 67 on the Reader Service Card

The Macro Assembler

The code I'll be presenting will generally be written in Assembler. I think C is a good development environment, but I'm in general agreement with Hal Hardenbergh's assertion that the best code (although not necessarily the easiest to write or the most reliable) is written in Assembler. This is especially true of graphics code for the 8086 family, given segments and the string instructions, and for real-time programming of a complex board like the EGA, there's really no other choice.

Before I'm deluged with protests from C devotees, let me add that the majority of my productive work is done in C; no programmer is immune to the laws of time, and C is simply a faster environment in which to develop, particularly when working in a programming team. I'd like to emphasize that there are many excellent C compilers available for the PC, some of which produce remarkably good code, and all of which are a damn sight more reliable and easier to develop with than the Microsoft Macro Assembler. In fact, I think MASM is a serious impediment to assembly-language programming on a computer that is inherently hard to program in Assembler. My experience with MASM began with version 1.0 compiling a DB that had a DUP factor of 0 into a block of 64 apparently random bytes, and matters have continued in pretty much the same vein ever since. Version 4.0 is much faster and has fewer bugs, but it's far from perfect. For one thing, macros don't

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

always work the way they should, and it can be very difficult just to pin down the source of an error in a nested macro, let alone devise a workaround. For another, the whole structure of the assembly language is, to put it charitably, odd. I've never seen another strongly-typed assembler before, and I hope I never do. I've also never seen another assembler capable of generating a phase error, surely one of the most maddening and least useful error messages ever. I use STRUC and GROUP and OFFSET, and I know a slew of tricks like using LEA or DGROUP:OFFSET to get an accurate offset in a group, but is all of this really necessary? In my opinion, MASM is somewhere between a true assembler and a high-level language, with most of the disadvantages of both.

One example, and then I'll get to the point. When a structure element is used in C, the compiler knows which structure definition that element is associated with and consequently has no trouble with elements of different structures that have the same names. For instance, LinkedListBlock.Status and DescriptorBlock.Status can coexist happily in the same module. Not so in Assembler—structure element names seem to simply translate into symbols, just as if an EQU had been performed, and the second structure element with a given name produces a redefinition error. As a result, I find myself naming elements like LLBStatus and DBStatus, a far cry from the power of C structures.

So what's the point? Well, I'm willing to bet that someone out there knows a way around the STRUC problem I've just described. Perhaps it's not a problem at all, and I've simply misunderstood the Macro Assembler manual, or perhaps there's a good workaround. Whether this particular problem is solvable or not, there are many tricks being used in similar situations to wring utility out of MASM. Like it or not, MASM is pretty much it for assembly-language development for the PC family. True, there are other assemblers, but all my existing code and most of the tools and high-level languages I use are designed to work with MASM, and every time I try to leave MASM I run into incompatibilities that send me running back to the old standard. Since MASM is so important, I'll be writing about the MASM tricks I know, and I'd appreciate it if you'd write in and share your MASM experiences with *PJ*'s readers as well.

Two unsolved MASM puzzles to start things off:

(1) I've never figured out how to use STRUC to define negative offsets from BP in a stack frame. C routinely addresses passed parameters at positive offsets from BP and dynamic storage at negative offsets, but since STRUC seems to generate only positive offsets, I've had to put dynamic storage at positive offsets and set BP only after dynamic storage has been allocated. Although this works, it effectively halves the number of bytes that can be addressed with single-byte offsets from BP, potentially increasing code size and execution time.

(2) While developing ROMable code, I needed to force certain code to be at a certain offset (say offset 2000h) in the final COM file. My thought was to use something like

```
DB      (2000h-$) DUP 0
```

to force the code to assemble at the desired location. Alas, the assembler can handle only a constant in this context, and "$" is not a constant. I tried a variety of other approaches, such as ORGing to the offset, but couldn't get the assembler to produce the desired result. I ended up disassembling the file, calculating the needed offset, and hardwiring it into a DB, which was fine until the next time I changed the code and forgot to adjust the DUP value in the DB. Does anyone know how to force code to assemble at an absolute location?

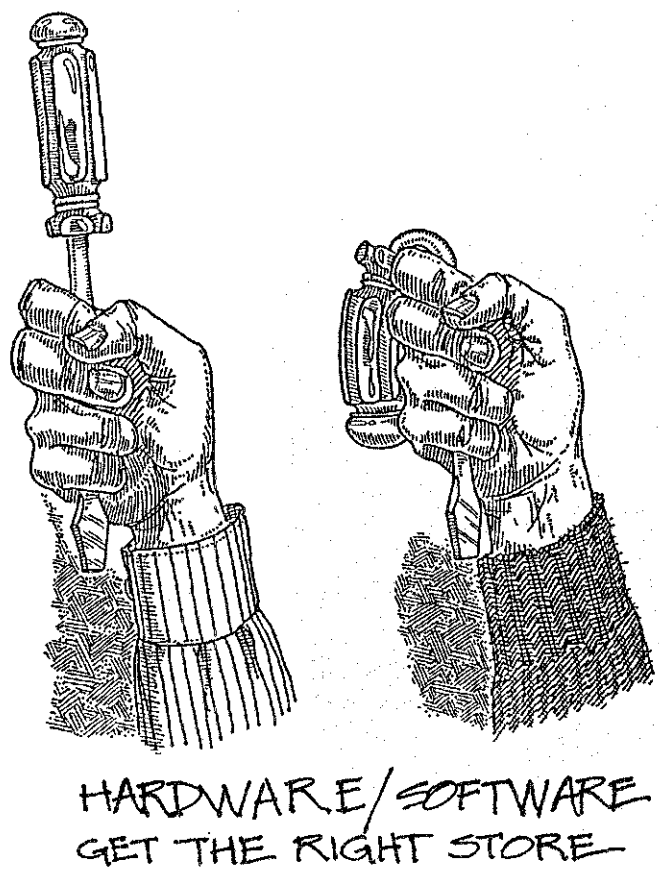
So. If you know the answers, let us know. If you've got bugs and workarounds of your own, let us know. We MASM developers can use all the help we can get.

Coming Up

Next time I'll look into the hardware assistance the EGA provides the CPU during display memory access. There are four latches and four ALUs in those chips, along with some useful masks and comparators; it should be interesting. See you then. □

Michael Abrash is a Senior Software Engineer for Orion Industries of Redwood City, CA, a manufacturer of PC-based instrumentation and microprocessor development systems.

Code follows.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
; Sample EGA program.
; Animates four balls bouncing around a playfield by using
; page flipping. Playfield is panned smoothly both horizontally
; and vertically.
; Assembled with MASM 4.0, linked with LINK 3.51.
; By Michael Abrash, 11/2/86.
;
stack   segment para stack 'STACK'
        db      512 dup(?)
stack   ends
;
HIRES_VIDEO_MODE        equ     0       ;define for 640x350 video mode
                                        ; comment out for 640x200 mode
VIDEO_SEGMENT   equ     0a000h          ;display memory segment for
                                        ; true EGA graphics modes
LOGICAL_SCREEN_WIDTH    equ     672/8   ;width in bytes and height in scan
LOGICAL_SCREEN_HEIGHT   equ     384     ; lines of the virtual screen
                                        ; we'll work with
PAGE0           equ     0       ;flag for page 0 when page flipping
PAGE1           equ     1       ;flag for page 1 when page flipping
PAGE0_OFFSET    equ     0       ;start offset of page 0 in EGA memory
PAGE1_OFFSET    equ     LOGICAL_SCREEN_WIDTH * LOGICAL_SCREEN_HEIGHT
                                ;start offset of page 1 (both pages
                                ; are 672x384 virtual screens)
BALL_WIDTH      equ     24/8    ;width of ball in display memory bytes
BALL_HEIGHT     equ     24      ;height of ball in scan lines
BLANK_OFFSET    equ     PAGE1_OFFSET * 2        ;start of blank image
                                                ; in EGA memory
BALL_OFFSET     equ     BLANK_OFFSET + (BALL_WIDTH * BALL_HEIGHT)
                                ;start offset of ball image in EGA memory
NUM_BALLS       equ     4       ;number of balls to animate
;
; EGA register equates.
;
TS_INDEX        equ     3c4h    ;TS index register
MAP_MASK        equ     2       ;TS map mask register
GDC_INDEX       equ     3ceh    ;GDC index register
GDC_MODE        equ     5       ;GDC mode register
CRTC_INDEX      equ     03d4h   ;CRTC index register
START_ADDRESS_HIGH equ  0ch     ;CRTC start address high byte
START_ADDRESS_LOW equ   0dh     ;CRTC start address low byte
CRTC_OFFSET     equ     13h     ;CRTC offset register
STATUS_REGISTER_1 equ   03dah   ;EGA status register
VSYNC_MASK      equ     08h     ;vertical sync bit in status register 1
ATC_INDEX       equ     03c0h   ;ATC index register
HPELPAN         equ     20h OR 13h      ;ATC horizontal pel panning register
                                        ; (bit 7 is high to keep palette RAM
                                        ; addressing on)
dseg    segment para common 'DATA'
CurrentPage             db      PAGE1           ;page to draw to
CurrentPageOffset       dw      PAGE1_OFFSET
;
; Four plane's worth of multicolored ball image.
;
BallPlane0Image label   byte            ;blue plane image
        db      000h, 03ch, 000h, 001h, 0ffh, 080h
        db      007h, 0ffh, 0e0h, 00fh, 0ffh, 0f0h
        db      4 * 3 dup(000h)
        db      07fh, 0ffh, 0feh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 0ffh, 0ffh, 0ffh
        db      4 * 3 dup(000h)
        db      07fh, 0ffh, 0feh, 03fh, 0ffh, 0fch
        db      03fh, 0ffh, 0fch, 01fh, 0ffh, 0f8h
        db      4 * 3 dup(000h)
BallPlane1Image label   byte            ;green plane image
        db      4 * 3 dup(000h)
        db      01fh, 0ffh, 0f8h, 03fh, 0ffh, 0fch
        db      03fh, 0ffh, 0fch, 07fh, 0ffh, 0feh
        db      07fh, 0ffh, 0feh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 0ffh, 0ffh, 0ffh
        db      8 * 3 dup(000h)
        db      00fh, 0ffh, 0f0h, 007h, 0ffh, 0e0h
```



```
        db      001h, 0ffh, 080h, 000h, 03ch, 000h
BallPlane2Image label   byte            ;red plane image
        db      12 * 3 dup(000h)
        db      0ffh, 0ffh, 0ffh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 07fh, 0ffh, 0feh
        db      07fh, 0ffh, 0feh, 03fh, 0ffh, 0fch
        db      03fh, 0ffh, 0fch, 01fh, 0ffh, 0f8h
        db      00fh, 0ffh, 0f0h, 007h, 0ffh, 0e0h
        db      001h, 0ffh, 080h, 000h, 03ch, 000h
BallPlane3Image label   byte            ;intensity on for all planes,
                                        ; to produce high-intensity colors
        db      000h, 03ch, 000h, 001h, 0ffh, 080h
        db      007h, 0ffh, 0e0h, 00fh, 0ffh, 0f0h
        db      01fh, 0ffh, 0f8h, 03fh, 0ffh, 0fch
        db      03fh, 0ffh, 0fch, 07fh, 0ffh, 0feh
        db      07fh, 0ffh, 0feh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 0ffh, 0ffh, 0ffh
        db      0ffh, 0ffh, 0ffh, 07fh, 0ffh, 0feh
        db      07fh, 0ffh, 0feh, 03fh, 0ffh, 0fch
        db      03fh, 0ffh, 0fch, 01fh, 0ffh, 0f8h
        db      00fh, 0ffh, 0f0h, 007h, 0ffh, 0e0h
        db      001h, 0ffh, 080h, 000h, 03ch, 000h
;
BallX           dw      15, 50, 40, 70          ;array of ball x coords
BallY           dw      40, 200, 110, 300       ;array of ball y coords
LastBallX       dw      15, 50, 40, 70          ;previous ball x coords
LastBallY       dw      40, 100, 160, 30        ;previous ball y coords
BallXInc        dw      1, 1, 1, 1              ;x move factors for ball
BallYInc        dw      8, 8, 8, 8              ;y move factors for ball
BallRep         dw      1, 1, 1, 1              ;# times to keep moving
                                                ; ball according to current
                                                ; increments
BallControl     dw      Ball0Control, Ball1Control      ;pointers to current
                dw      Ball2Control, Ball3Control      ; locations in ball
                                                        ; control strings
BallControlString       dw      Ball0Control, Ball1Control ;pointers to
                        dw      Ball2Control, Ball3Control ; start of ball
                                                           ; control strings
;
; Ball control strings.
;
Ball0Control    label   word
        dw      10, 1, 4, 10, -1, 4, 10, -1, -4, 10, 1, -4, 0
Ball1Control    label   word
        dw      12, -1, 1, 28, -1, -1, 12, 1, -1, 28, 1, 1, 0
Ball2Control    label   word
        dw      20, 0, -1, 40, 0, 1, 20, 0, -1, 0
Ball3Control    label   word
        dw      8, 1, 0, 52, -1, 0, 44, 1, 0, 0
;
; Panning control string.
;
ifdef HIRES_VIDEO_MODE
PanningControlString    dw      32, 1, 0, 34, 0, 1, 32, -1, 0, 34, 0, -1, 0
else
PanningControlString    dw      32, 1, 0, 184, 0, 1, 32, -1, 0, 184, 0, -1, 0
endif
PanningControl  dw      PanningControlString    ;pointer to current location
                                                ; in panning control string
PanningRep      dw      1       ;# times to pan according to current
                                ; panning increments
PanningXInc     dw      1       ;x panning factor
PanningYInc     dw      0       ;y panning factor
HPan            db      0       ;horizontal pel panning setting
PanningStartOffset dw   0       ;start offset adjustment to produce vertical
                                ; panning & coarse horizontal panning
dseg    ends
;
; Macro to set indexed register P2 of chip with index register
; at P1 to AL.
;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code cont.

3

```
SETREG  macro   P1, P2
        mov     dx,P1
        mov     ah,al
        mov     al,P2
        out     dx,ax
        endm
;
cseg    segment para public 'CODE'
        assume  cs:cseg, ds:dseg
start   proc    near
        mov     ax,dseg
        mov     ds,ax
;
; Select graphics mode.
;
ifdef HIRES_VIDEO_MODE
        mov     ax,010h
else
        mov     ax,0eh
endif
        int     10h
;
; ES always points to EGA memory.
;
        mov     ax,VIDEO_SEGMENT
        mov     es,ax
;
; Draw border around playfield in both pages.
;
        mov     di,PAGE0_OFFSET
        call    DrawBorder      ;page 0 border
        mov     di,PAGE1_OFFSET
        call    DrawBorder      ;page 1 border
;
; Draw all four plane's worth of the ball to undisplayed EGA memory.
        mov     al,01h          ;enable plane 0
        SETREG  TS_INDEX, MAP_MASK
        mov     si,offset BallPlane0Image
        mov     di,BALL_OFFSET
        mov     cx,BALL_WIDTH * BALL_HEIGHT
        rep movsb
        mov     al,02h          ;enable plane 1
        SETREG  TS_INDEX, MAP_MASK
        mov     si,offset BallPlane1Image
        mov     di,BALL_OFFSET
        mov     cx,BALL_WIDTH * BALL_HEIGHT
        rep movsb
        mov     al,04h          ;enable plane 2
        SETREG  TS_INDEX, MAP_MASK
        mov     si,offset BallPlane2Image
        mov     di,BALL_OFFSET
        mov     cx,BALL_WIDTH * BALL_HEIGHT
        rep movsb
        mov     al,08h          ;enable plane 3
        SETREG  TS_INDEX, MAP_MASK
        mov     si,offset BallPlane3Image
        mov     di,BALL_OFFSET
        mov     cx,BALL_WIDTH * BALL_HEIGHT
        rep movsb
;
; Draw a blank image the size of the ball to undisplayed EGA memory.
;
        mov     al,0fh                  ;enable all memory planes, since the
        SETREG  TS_INDEX, MAP_MASK      ; blank has to erase all planes
        mov     di,BLANK_OFFSET
        mov     cx,BALL_WIDTH * BALL_HEIGHT
        sub     al,al
        rep stosb
;
; Set EGA to write mode 1, for block copying ball and blank images.
;
        mov     al,1
        SETREG  GDC_INDEX, GDC_MODE
;
```

4

```
; Set EGA offset register in words to define logical screen width.
;
        mov     al,LOGICAL_SCREEN_WIDTH / 2
        SETREG  CRTC_INDEX, CRTC_OFFSET
;
; Move the balls by erasing each ball, moving it, and
; redrawing it, then switching pages when they're all moved.
;
BallAnimationLoop:
        mov     bx,( NUM_BALLS * 2 ) - 2
EachBallLoop:
;
; Erase old image of ball in this page (at location from one more earlier).
;
        mov     si,BLANK_OFFSET ;point to blank image
        mov     cx,[LastBallX+bx]
        mov     dx,[LastBallY+bx]
        call    DrawBall
;
; Set new last ball location.
;
        mov     ax,[BallX+bx]
        mov     [LastballX+bx],ax
        mov     ax,[BallY+bx]
        mov     [LastballY+bx],ax
;
; Change the ball movement values if it's time to do so.
;
        dec     [BallRep+bx]            ;has current repeat factor run out?
        jnz     MoveBall
        mov     si,[BallControl+bx]     ;it's time to change movement values
        lodsw                           ;get new repeat factor from
                                        ; control string
        and     ax,ax                   ;at end of control string?
        jnz     SetNewMove
        mov     si,[BallControlString+bx]       ;reset control string
        lodsw                           ;get new repeat factor
SetNewMove:
        mov     [BallRep+bx],ax         ;set new movement repeat factor
        lodsw                           ;set new x movement increment
        mov     [BallXInc+bx],ax
        lodsw                           ;set new y movement increment
        mov     [BallYInc+bx],ax
        mov     [BallControl+bx],si     ;save new control string pointer
;
; Move the ball.
;
MoveBall:
        mov     ax,[BallXInc+bx]
        add     [BallX+bx],ax           ;move in x direction
        mov     ax,[BallYInc+bx]
        add     [BallY+bx],ax           ;move in y direction
;
; Draw ball at new location.
;
        mov     si,BALL_OFFSET  ;point to ball's image
        mov     cx,[BallX+bx]
        mov     dx,[BallY+bx]
        call    DrawBall
;
        dec     bx
        dec     bx
        jns     EachBallLoop
;
; Pan and flip displayed page to the one we've been working on, as soon
; as leading edge of vertical sync comes around.
;
        call    WaitVSync
;
; Perform whatever panning's in effect.
;
        call    AdjustPanning
;
; Flip to new page by changing start address.
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
        mov     ax,[CurrentPageOffset]
        add     ax,[PanningStartOffset]
        push    ax
        SETREG  CRTC_INDEX, START_ADDRESS_LOW
        mov     al,byte ptr [CurrentPageOffset+1]
        pop     ax
        mov     al,ah
        SETREG  CRTC_INDEX, START_ADDRESS_HIGH
;
; Wait for sync again so the new start address has a chance
; to take effect.
;
        call    WaitVSync
;
; Set horizontal panning now, just as new start address takes effect.
;
        mov     al,[HPan]
        mov     dx,STATUS_REGISTER_1
        in      al,dx                   ;reset ATC addressing to index reg
        mov     al,[HPan]
        SETREG  ATC_INDEX, HPELPAN
;
; Flip the page to draw to to the undisplayed page.
;
        xor     [CurrentPage],1
        jnz     IsPage1
        mov     [CurrentPageOffset],PAGE0_OFFSET
        jmp     short EndFlipPage
IsPage1:
        mov     [CurrentPageOffset],PAGE1_OFFSET
EndFlipPage:
;
; Exit if a key's been hit.
;
        mov     ah,1
        int     16h
        jnz     Done
        jmp     BallAnimationLoop
;
; Finished, clear key, reset screen mode and exit.
;
Done:
        mov     ah,0    ;clear key
        int     16h
;
        mov     ax,3    ;reset to text mode
        int     10h
;
        mov     ah,4ch  ;exit to DOS
        int     21h
;
start   endp
;
; Routine to draw a ball-sized image to all planes, copying from
; offset SI in EGA memory to offset CX,DX (x,y) in EGA memory in
; the current page.
;
DrawBall        proc    near
        mov     ax,LOGICAL_SCREEN_WIDTH
        mul     dx      ;offset of start of top image scan line
        add     ax,cx   ;offset of upper left of image
        add     ax,[CurrentPageOffset]  ;offset of start of page
        mov     di,ax
        mov     bp,BALL_HEIGHT
        push    ds
        push    es
        pop     ds      ;move from EGA memory to EGA memory
DrawBallLoop:
        push    di
        mov     cx,BALL_WIDTH
        rep movsb       ;draw a scan line of image
        pop     di
        add     di,LOGICAL_SCREEN_WIDTH ;point to next destination scan line
```



```
        dec     bp
        jnz     DrawBallLoop
        pop     ds
        ret
DrawBall        endp
;
; Wait for the leading edge of vertical sync pulse.
;
WaitVSync       proc    near
        mov     dx,STATUS_REGISTER_1
WaitNotVSyncLoop:
        in      al,dx
        and     al,VSYNC_MASK
        jnz     WaitNotVSyncLoop
WaitVSyncLoop:
        in      al,dx
        and     al,VSYNC_MASK
        jz      WaitVSyncLoop
        ret
WaitVSync       endp
;
; Perform horizontal/vertical panning.
;
AdjustPanning   proc    near
        dec     [PanningRep]    ;time to get new panning values?
        jnz     DoPan
        mov     si,[PanningControl]     ;point to current location in
                                        ; panning control string
        lodsw                           ;get panning repeat factor
        and     ax,ax                   ;at end of panning control string?
        jnz     SetnewPanValues
        mov     si,offset PanningControlString  ;reset to start of string
        lodsw                           ;get panning repeat factor
SetNewPanValues:
        mov     [PanningRep],ax         ;set new panning repeat value
        lodsw
        mov     [PanningXInc],ax        ;horizontal panning value
        lodsw
        mov     [PanningYInc],ax        ;vertical panning value
        mov     [PanningControl],si     ;save current location in panning
                                        ; control string
;
; Pan according to panning values.
;
DoPan:
        mov     ax,[PanningXInc]        ;horizontal panning
        and     ax,ax
        js      PanLeft                 ;negative means pan left
        jz      CheckVerticalPan
        mov     al,[HPan]
        inc     al                      ;pan right; if pel pan reaches
        cmp     al,8                    ; 8, it's time to move to the
        jb      SetHPan                 ; next byte with a pel pan of 0
        sub     al,al                   ; and a start offset that's one
        inc     [PanningStartOffsat]    ; higher
        jmp     short SetHPan
PanLeft:
        mov     al,[HPan]
        dec     al                      ;pan left; if pel pan reaches -1,
        jns     SetHPan                 ; it's time to move to the next
        mov     al,7                    ; byte with a pel pan of 7 and a
        dec     [PanningStartOffset]    ; start offset that's one lower
SetHPan:
        mov     [HPan],al               ;save new pel pan value
CheckVerticalPan:
        mov     ax,[PanningYInc]        ;vertical panning
        and     ax,ax
        js      PanUp                   ;negative means pan up
        jz      EndPan
        add     [PanningStartOffset],LOGICAL_SCREEN_WIDTH
                                        ;pan down by advancing the start
                                        ; address by a scan line
        jmp     short EndPan
PanUp:
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code cont.

7

```
        sub     [PanningStartOffset],LOGICAL_SCREEN_WIDTH
                                ;pan up by retarding the start
                                ; address by a scan line
EndPan:
        ret

;
; Draw textured border around playfield that starts at DI.
;
DrawBorder      proc    near
;
; Draw the left border.
;
        push    di
        mov     cx,LOGICAL_SCREEN_HEIGHT / 16
DrawLeftBorderLoop:
        mov     al,0ch          ;select red color for block
        call    DrawBorderBlock
        add     di,LOGICAL_SCREEN_WIDTH * 8
        mov     al,0eh          ;select yellow color for block
        call    DrawBorderBlock
        add     di,LOGICAL_SCREEN_WIDTH * 8
        loop    DrawLeftBorderLoop
        pop     di
;
; Draw the left border.
;
        push    di
        add     di,LOGICAL_SCREEN_WIDTH - 1
        mov     cx,LOGICAL_SCREEN_HEIGHT / 16
DrawRightBorderLoop:
        mov     al,0eh          ;select yellow color for block
        call    DrawBorderBlock
        add     di,LOGICAL_SCREEN_WIDTH * 8
        mov     al,0ch          ;select red color for block
        call    DrawBorderBlock
        add     di,LOGICAL_SCREEN_WIDTH * 8
        loop    DrawRightBorderLoop
        pop     di
;
; Draw the top border.
;
        push    di
        mov     cx,(LOGICAL_SCREEN_WIDTH - 2) / 2
DrawTopBorderLoop:
        inc     di
        mov     al,0eh          ;select yellow color for block
        call    DrawBorderBlock
        inc     di
        mov     al,0ch          ;select red color for block
        call    DrawBorderBlock
        loop    DrawTopBorderLoop
        pop     di
;
; Draw the bottom border.
;
        add     di,(LOGICAL_SCREEN_HEIGHT - 8) * LOGICAL_SCREEN_WIDTH
        mov     cx,(LOGICAL_SCREEN_WIDTH - 2) / 2
DrawBottomBorderLoop:
        inc     di
        mov     al,0ch          ;select red color for block
        call    DrawBorderBlock
        inc     di
        mov     al,0eh          ;select yellow color for block
        call    DrawBorderBlock
        loop    DrawBottomBorderLoop
        ret
DrawBorder      endp
;
; Draws an 8x8 border block in color in AL at location DI.
; DI preserved.
;
```

8

```
DrawBorderBlock proc    near
        push    di
        SETREG  TS_INDEX, MAP_MASK
        mov     al,0ffh
        rept 8
        stosb
        add     di,LOGICAL_SCREEN_WIDTH - 1
        endm
        pop     di
        ret
DrawBorderBlock endp
AdjustPanning   endp
cseg    ends
        end     start
```

WRITE IT DOWN

Send us your article or review on disk — standard ASCII text file, please — accompanied by a printout. Attach some information about yourself if you'd like that to appear, too. Expect to hear from us about your article within the month. Send us a stamped self-addressed envelope if you'd like us to return your disk and manuscript.

In addition to the satisfaction and prestige to be gotten from publishing in *Programmer's Journal*, you can make money, too — we won't make you rich, but we can help make you famous. Think about it . . . Write about it!

Nobody knows better than the readers of *Programmer's Journal* exactly the kinds of information, tips, and tricks that programmers can use. We want to hear what you have to say about code you've developed, products you've tried, the language(s) you favor, books you've read. We're interested in your views on general computing topics, too — artificial intelligence, commercial development, legal questions, the latest computer conference you attended — the way things are shaping up for programmers. Write it down and send it to

Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Version 3.0

New Features

Windows, Data Entry, Help Management, Menus, Text Editing, plus ...

SOURCE CODE

Vitamin C

It's good for your system!

The Vitamin C Difference

With **Vitamin C**, your applications come alive with windows that explode into view! Data entry windows and menus become a snap, and context sensitive pop-up help messages are nearly automatic.

With **VCScreen,** you'll save time by interactively painting windows and forms so what you see is what you get! Then, one button generates C source code ready to plug into your program and link with Vitamin C.

Easy enough for the beginner. Versatile enough for the professional. Vitamin C's **open-ended design** is full of "hooks" so you can intercept and "plug-in" special handlers to customize or add features to most routines.

Of course, Vitamin C **includes all source code FREE,** with no hidden charges. *It always has.* That means you'll have everything you need to adapt to special needs without spending hundreds of dollars more.

Windows

Create as many windows as you like with one easy function. Vitamin C automatically takes care of complicated tasks like saving and restoring the area under a window.

Options include titles, borders, colors, pop-up, pull-down, zoom-in, 4-way scrolling, scroll bars, sizes up to 32k, text file display & editing, cursor display, and more.

Unique built-in feature lets users move and resize windows during run-time via a definable key.

Access the current window by default or a specific window any time, even if it's hidden or invisible. Save and load windows on disk for more versatility!

Data Entry

Flexible dBase-like data entry and display routines feature protected, invisible, required, and scrolling fields. Picture clause formatting, full color/attribute control, selection sets, single field and full screen input, and unlimited data validation via standard and user definable routines. That means you aren't locked into one way of doing things.

Vitamin C even provides true right-to-left input of numeric fields with dynamic display of separators & currency symbols.

High Level Functions

Use our intergrated help management, multi-level menus, and text file routines, or build your own handlers using Vitamin C's basic windowing and data entry routines.

Standard help handler provides context sensitive pop-up help messages any time the program awaits key strokes. The help text file is stored on disk and indexed for quick access. So easy to use that a single function initializes & services requests by opening a window, locating, formatting, displaying, and paging through the message.

Multi-level "MacIntosh" & "Lotus" style menus make user interfaces and front ends a snap. Menus can call other menus, functions, even data entry screens, quickly and easily.

Text editor windows can be opened for pop-up note pads, memo fields, or general purpose editing. Features include insert, delete, word wrap, and paragraph formatting.

VCScreen

Screen Painter/Code Generator

Just as Vitamin C's reusable functions speed your programming, VCSreen makes it even *faster and easier* by automatically generating C source code for your data entry screens!

With VCScreen's interactive screen editor, you actually draw your forms. You can define input, output and constant fields, headings, boxes, lines and even a window for the form to run in.

What you see is what you get. If you don't like the position of an object, just "pick it up" with the cursor and move it! Changing colors, attributes, copying, and deleting is just as easy.

VCScreen generates readable C source code. It declares variables with names you provide and can even generate structures.

With VCScreen choosing the right functions, parameters and sequences, and Vitamin C supplying the functions to choose from, you can stop worrying about semi-colons, matching braces, and calling conventions and concentrate on creating your application!

30 Day Money Back Guarantee

Better than a brochure. More than a demo disk. If you're not satisfied, simply return the package within 30 days and receive a full refund of the purchase price.

Vitamin C $225.00

Includes ready to use libraries, tutorial, reference manual, demo, sample, and example programs, and quick reference card. For IBM PC and compatibles. Specify Microsoft, Lattice, Computer Innovations, Aztec, Mark Williams, Wizard, DeSmet, or Datalight C compiler AND compiler version number when ordering.

Vitamin C Source ... FREE*

**Free with purchase of Vitamin C*

VCScreen $99.95

Requires Vitamin C and IBM PC/XT/AT or true compatible.

ALL ORDERS:

SHIPPING: $3 ground, $6 2-day air, $20 overnight, $30 overseas. Visa and Master Card accepted. All funds must be U.S. dollars drawn on a U.S. Bank. Texas residents add 6⅛% sales tax.

For Orders or More Information, Call ...

(214) 245-6090

creative PROGRAMMING

Creative Programming Consultants, Inc.
Box 112097 Carrollton, Texas 75011

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

How Many Holes Does It Take to Fill an IBM PC?

by
Tom Swan

Strings in Pascal eat memory the way black holes in space eat light. Once light disappears into a black hole's gullet, it never again is seen. And once memory disappears into a string variable, it's gone for good—no other variable can use it.

String variables are wasteful because they occupy fixed amounts of memory. An 80-character string takes 81 bytes (the extra byte stores the string length) no matter how many characters it actually has. Even an empty or null string takes as much room as a full one.

And string arrays are even worse. As much as 50 percent or more of a 100-element string array often goes to waste unless, of course, you completely fill every string in the array to capacity.

One way to eliminate such waste is to declare pointers to strings rather than directly declare the variables themselves. By doing this, you create variable-length strings that occupy only as much memory as they need. As two simple examples show, this once-removed approach to managing string space helps plug up wasteful holes in memory.

Plain String Arrays

The first example program, STRING1.PAS, demonstrates the usual way to read text files into a string array. You might use a similar method to read and display "help" text, to page through on-line instruction manuals, or as the basis of a text processing program.

Constant MaxString determines the maximum number of strings that TYPE StrArray holds. Type Str255 defines a maximum-length string, which occupies 256 bytes. Therefore, array variable sa takes 100x256 or 25,600 bytes of memory.

The program requests the name of a text file. It then reads and displays lines from that file reading and writing MaxStrings lines at a time. Procedure ReadStrings reads text lines into arrays sa and keeps account in variable saCount of the number of strings in the array.

Procedure WriteStrings displays saCount strings—just to let you know the program is working. Procedure AverageMemory keeps a running average of the amount of available memory. It does this with the help of Turbo's MemAvail function, which returns the number of available 16-byte paragraphs on the Pascal heap. This represents the total amount of memory now unused. Our goal is to increase this value as much as possible and, therefore, increase the amount of memory available for other purposes.

After displaying all lines from the file, the program reports the final average of available memory. When you run it, you'll notice that this value never changes. No matter how big your text file and, even more important, no matter how long or how short its lines, the amount of available memory remains fixed. The average on my XT was 4,976 paragraphs or 79,616 bytes.

Obviously, STRING1.PAS wastes a good deal of memory. Every text file blank line wastes a 256 byte hole that you cannot use for anything else. The only way STRING1.PAS would not be so inefficient is if every line were exactly 255 characters long, an unlikely prospect.

Plugging the Holes

A second program, STRING2.PAS, shows one way to reduce the size of the string array and, therefore, increase the amount of available memory. Much of the new program duplicates the first example but, although the physical differences are subtle, the results are dramatic.

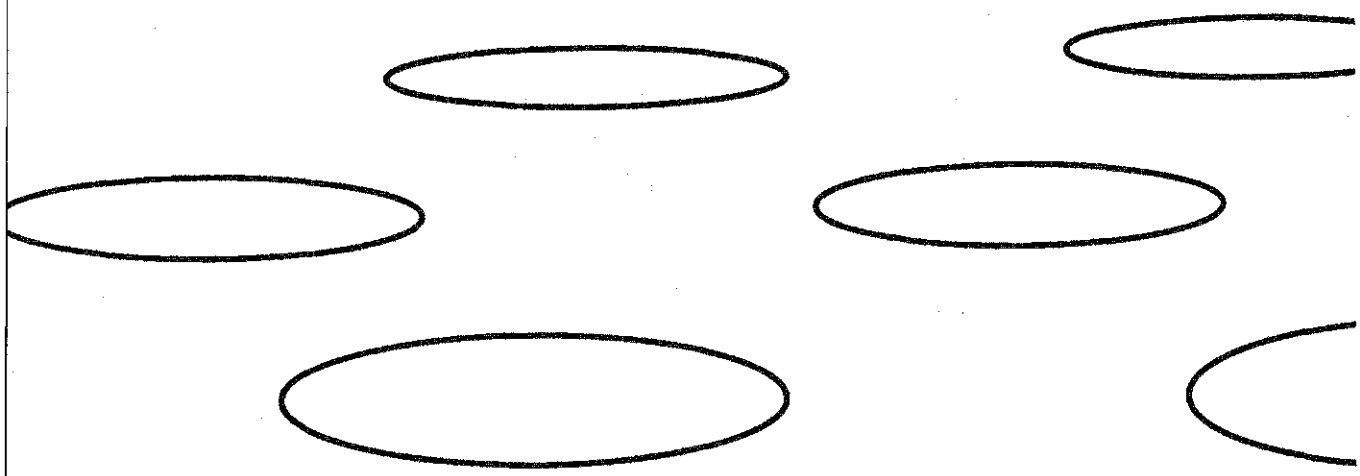
StrArray is now an array of string pointers, rather than of string variables. The caret in front of ^Str255 in the array declaration means "a pointer to." Instead of storing fixed length strings in the array, it now holds string pointers. Because pointers take four bytes (two for the segment and two for the offset addresses), the new array occupies only 400 instead of 25,600 bytes.

Procedure WriteStrings is nearly identical to the same procedure in STRING1.PAS except for the caret after sa[j]^, an example of a "dereferenced pointer." Without the caret, sa[j] is merely one array element—in this case, a pointer. But with the caret, sa[j]^ means "the thing to which the pointer points." The caret tells the compiler to interpret sa[j]^ as the object at the location of the pointer. The compiler considers the type of that object the same as the data type you originally used in the pointer's declaration—in this example, the string type Str255.

Procedure AverageMemory and the main program loop are unchanged. Procedure Initialize is almost identical but adds the statement:

```
Mark( heap )
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



which records the address of the top of the Pascal memory heap. Variable heap could be a pointer to any data type but traditionally is a pointer to an INTEGER. The program needs to mark the heap location because another procedure, ReadStrings, will place strings onto the heap allocating memory as needed for each new string. By recording the original location of the heap, and resetting it later, the program recovers memory occupied by strings for which it no longer has any use.

This "mark and release" method is the simplest memory managing technique you can use in Pascal. The heap expands to hold new variables, then shrinks down to its original size, releasing those variables and recovering the memory for other purposes. Measuring the amount of that memory should tell us whether the new program reduces waste.

The New ReadStrings Procedure

In the statement Release(heap), the new ReadStrings resets the heap to the value recorded by Mark(heap) in procedure Initialize. This throws away any strings now in memory and recovers that memory for use by new strings.

As in the first example, the procedure sets saCount to the number of strings it reads. Unlike the first example, the new procedure reads text into a temporary string variable, s. After that, it allocates just enough memory to hold the characters in the string. It does this with the statement:

```
GetMem( sa[saCount], Length(s)+1 );
```

The statement reserves Length(s)+1 bytes on the heap and assigns the address of the first byte of that memory to pointer sa[saCount]. Because every Pascal string begins with a byte that indicates how many characters are in the string, the program reserves Length(s)+1 bytes—exactly enough room for every character in the string plus one byte for the string length value.

Finally, ReadStrings assigns the temporary string to sa[saCount]^. The caret, remember, dereferences the pointer making the compiler recognize sa[saCount]^ as a string variable of type Str255. It therefore allows the assignment because string s is also of type Str255.

This new program usually reports a much higher average of available memory than does the first example. Also, the average is likely to change depending on whether your text files have many long or many short strings. On my XT, running STRING2 on a variety of files—mostly Pascal programs, which typically have many short strings—reported an average of 6,400 available paragraphs compared to STRING1's 4976, a savings of more than 22,500 bytes. Also, informal time trials indicate that this savings costs little if anything in program speed. Despite the extra assignment to the temporary string variable, both examples run at similar rates. Still, even though these are good results, there are disadvantages that you should carefully consider before using variable-length strings in your own programs.

String Pointer Dangers

Using string pointers instead of variables, and using GetMem to allocate only as much space as a string needs, is not without its dangers. If you have a string pointer SP of type ^Str255, GetMem lets you create a simple 20-byte variable string with the statement:

```
GetMem( SP, 20 );
```

But Pascal still believes that SP points to a 255-character string. Because of that belief, the compiler allows literal assignments of 80-character or longer strings to SP^ although you reserved room for only 19 characters. (Remember that extra length byte!)

One way to stay out of danger is to never increase the length of a variable-length string. You can safely decrease its length by assigning shorter strings or by using one of Pascal's string procedures such as Delete. But, you can never increase its length beyond the reserved memory without overwriting whatever variable comes after the string. There is no way to force Pascal to check for such errors. You have to do it yourself.

To help prevent string-length errors, create a record with fields for the string pointer and a maximum length value. You might use a data type like this:

```
TYPE
   StrRec=
      RECORD
         TheLength : INTEGER;
         ThePointer : ^Str255
      END;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Managing an array of these records takes a bit more work. After the ReadLn statement in ReadStrings, you could allocate and assign new strings in a WITH statement. Assume sa is an array of StrRec records.

```
WITH sa[saCount] DO
BEGIN
   TheLength := Length(s);
   GetMem( ThePointer, TheLength+1 )
   ThePointer^ := s
END;
```

You now have three things for each array element: the maximum string length TheLength; a pointer ThePointer to the string variable in memory; and the string itself, which you reference by writing sa[n].ThePointer^.

Because you know the string's maximum length, you can pass it to procedures such as ReadLn that might change its length. You still have to be careful not to exceed the string variable's size, but at least you now have the means to do that.

Code from Tom Swan's "How Many Holes Does It Take to Fill an IBM PC?"

1

```
PROGRAM String1;

{ Read and write strings using a plain string array }

CONST
  MaxString = 100;              { Size of string array }

TYPE
  Str255 = String[255];         { Maximum-length string }
  StrArray =
    ARRAY[ 1 .. MaxString ] OF Str255;   { Array of strings }

VAR
  f          : TEXT;            { Text file }
  fileName   : String[64];      { File or path name }
  sa         : StrArray;        { String array }
  saCount    : INTEGER;         { Number of strings in sa }
  average    : INTEGER;         { Average available memory }

PROCEDURE ReadStrings;
{ Read strings into global array }
BEGIN
  saCount := 0;
  WHILE ( saCount < MaxString ) AND ( NOT Eof( f ) ) DO
  BEGIN
    saCount := saCount + 1;
    Readln( f, sa[ saCount ] )
  END { while }
END; { ReadStrings }

PROCEDURE WriteStrings;
{ Display strings in sa array }
VAR
  j : INTEGER;
BEGIN
  FOR j := 1 TO saCount DO
    writeln( sa[j] )
END; { WriteStrings }

PROCEDURE AverageMemory;
{ Calculate running average of available memory }
BEGIN
  average := ( average + MemAvail ) DIV 2
END; { AverageMemory }

PROCEDURE Initialize;
{ 1. Open a text file }
{ 2. Initialize global variables }
BEGIN
  Write( 'File? ' );
  Readln( fileName );
  Assign( f, fileName );
  Reset( f );
  saCount := 0;
  average := MemAvail
END; { Initialize }

BEGIN
  Initialize;
  WHILE NOT Eof( f ) DO
  BEGIN
    ReadStrings;
    WriteStrings;
    AverageMemory
  END; { while }
  writeln;
  writeln( 'Average memory = ', average )
END.
```

2

```
PROGRAM String2;

{ Read and write strings using a string pointer array }

CONST
  MaxString = 100;              { Size of string pointer array }

TYPE
  Str255 = String[255];         { Maximum-length string }
  StrArray =
    ARRAY[ 1 .. MaxString ] OF ^Str255;   { Array of string pointers }

VAR
  f          : TEXT;            { Text file }
  fileName   : String[64];      { File or path name }
  sa         : StrArray;        { String pointer array }
  saCount    : INTEGER;         { Number of strings in sa }
  average    : INTEGER;         { Average available memory }
  heap       : ^INTEGER;        { Location of memory heap }

PROCEDURE ReadStrings;
{ Read strings into global array }
VAR
  s : Str255;
BEGIN
  Release( Heap );   { Erase strings now in memory }
  saCount := 0;
  WHILE ( saCount < MaxString ) AND ( NOT Eof( f ) ) DO
  BEGIN
    saCount := saCount + 1;
    Readln( f, s );
    Getmem( sa[ saCount ], Length( s ) + 1 );
    sa[ saCount ]^ := s
  END { while }
END; { ReadStrings }

PROCEDURE WriteStrings;
{ Display strings in sa array }
VAR
  j : INTEGER;
BEGIN
  FOR j := 1 TO saCount DO
    writeln( sa[j]^ )
END; { WriteStrings }

PROCEDURE AverageMemory;
{ Calculate running average of available memory }
BEGIN
  average := ( average + MemAvail ) DIV 2
END; { AverageMemory }

PROCEDURE Initialize;
{ 1. Open a text file }
{ 2. Initialize global variables }
BEGIN
  Write( 'File? ' );
  Readln( fileName );
  Assign( f, fileName );
  Reset( f );
  saCount := 0;
  average := MemAvail;
  Mark( heap )
END; { Initialize }

BEGIN
  Initialize;
  WHILE NOT Eof( f ) DO
  BEGIN
    ReadStrings;
    WriteStrings;
    AverageMemory
  END; { while }
  writeln;
  writeln( 'Average memory = ', average )
END.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

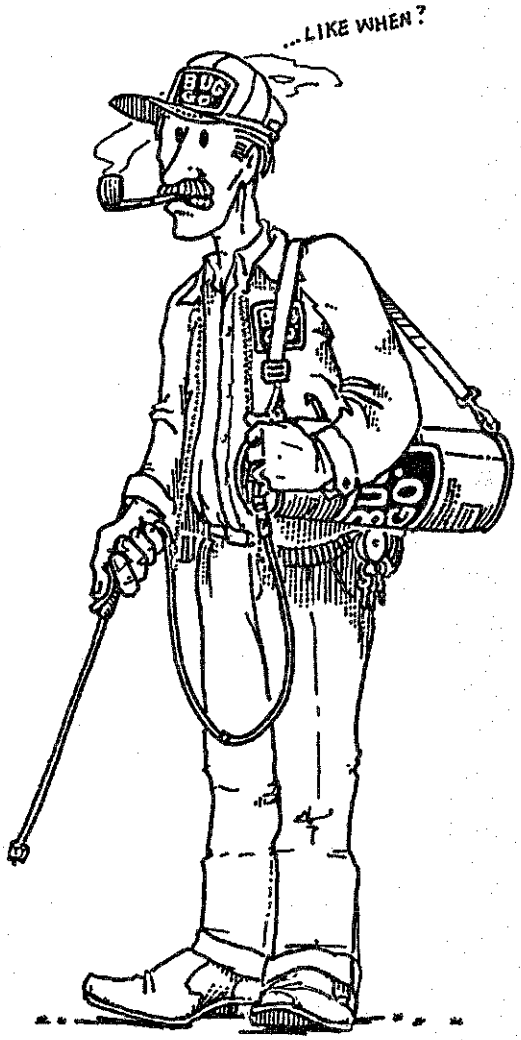
For example, to let someone type into a variable-length string, you could write:

```
WITH sa[n] DO
BEGIN
   BufLen := TheLength;
   ReadLn( ThePointer^ )
END;
```

The assignment to BufLen—probably unique to Turbo Pascal—tells ReadLn never to allow more than BufLen characters. Doing this makes it impossible to type more than TheLength characters into the string at ThePointer^.

Despite their potential dangers, variable-length strings reduce waste by plugging unused holes in memory. With the help of Pascal's pointers and Turbo's GetMem procedure, variable-length strings take only as much space as they need. This often produces dramatic memory savings with practically no loss of program speed. It's a technique that makes the extra programming effort worthwhile. □

Tom Swan is the author of several books including Pascal Programs for Data Base Management *and* Mastering Turbo Pascal *published by Howard W. Sams, Inc.*



Taskview™

WHY GIVE UP...

BATCH FILES,
I/O REDIRECTION,
SIDEKICK®,
DOS MENU PROGRAMS,
MOST OF YOUR RAM,
EXECUTION SPEED?

Compatible, efficient DOS multi-tasking.

We designed Taskview with efficiency in mind. During normal operation, TASKVIEW hides behind DOS, providing you with control of up to 10 concurrent or non-concurrent programs. Just the touch of a key instantly switches a program to the foreground. Included desktop utilities let you cut and paste from program to program. Simple to use and reasonably priced, no well equipped PC user should be without it.

Requires: PC/AT/Jr compatible, DOS 2.0-3.1, 256K RAM, 1 Floppy drive.

Taskview trademark of Sunnyhill Software
Sidekick registered trademark of Borland Intl

30-day money back guarantee

Dealer Inquiries Invited.

$69.95 plus $5.00 S&H

Washington residents add 7.9%
International orders add $5.00
VISA and Mastercard accepted.

To order Toll-Free
call 1-800-367-0651

Sunny Hill Software
13732 Midvale N. Ste. 206
Seattle, WA 98133
(206) 367-0650 M-F, 8-6 PDT

Number 85 on the Reader Service Card

Turbo Professional™

SERVICE INTERRUPTS
No assembly required
RESIDENT PROGRAMS
Easy, pop-up routines
EXECUTIVE PROGRAMS
Run ANY DOS program
DISK SECTOR I/O
Lowest level access
FAST TEXT WINDOWS
Virtual windowing system
KEYBOARD MACROS
Simple, powerful
LOTS OF EXAMPLES
21+ full example programs
MUCH MORE...
Over 140 routines in all

"If you never thought Turbo Pascal was a systems programming language, you've never seen Turbo Professional."
Darryl Rubin
Computer Language

For programs that move with technology—Turbo Professional—a truly professional library of subroutines.

150 page reference manual.
Full source—many example programs.
No royalties charged for applications.

Requires IBM compatible, DOS version 2.0 or greater, Turbo Pascal 2.0 or greater.

Turbo Professional, trademark of Sunnyhill Software
Turbo Pascal, registered trademark of Borland International

Dealer Inquiries Invited.

$69.95 plus $5.00 S&H

Washington residents add 7.9%
International orders add $5.00
VISA and Mastercard accepted.

To order Toll-Free
call 1-800-367-0651

Sunny Hill Software
13732 Midvale N. Ste. 206
Seattle, WA 98133
(206) 367-0650 M-F, 8-6 PDT

Number 85 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Proposed ANSI C Language Standard, Part 5

by
Rex Jaeschke

©*Rex Jaeschke, 1986. All rights reserved.*

Public Review Period Announced

On October 10th, 1986, the ANSI X3 Secretariat issued the following news release:

> *Washington, D.C.*—X3, the Accredited Standards Committee on Information Processing Systems, announces a four-month public review and comment period on draft proposed American National Standard, X3.159-198x. The public review period extends from November 7, 1986 to March 7, 1987.
>
> This standard specifies the form and establishes the interpretation of programs expressed in the programming language C. Its purpose is to promote portability, reliability, maintainability, and efficient execution of C language programs on a variety of computer systems. Sections are included that detail the C language itself and the contents of the C language execution library. Appendices summarize aspects of both of them and enumerate factors that influence the portability of C programs. While this standard is intended to guide knowledgeable C language programmers as well as implementers of C language translation systems, the document is not designed to serve as a tutorial.
>
> This draft standard is available for public review and comment for a four-month period ending March 7, 1987. Copies may be obtained from GLOBAL ENGINEERING DOCUMENTS, INC., by calling (800) 854-7179 or (714) 540-9870.
>
> Single copy price: US$65.00

Global Engineering are located in Santa Ana, California, USA, which is in the Pacific Standard Time (PST) zone.

For $65 you get both the draft Standard and the Rationale document. And it is almost certain there will be at least one two-month review period later in 1987, resulting in a further update to each document. Be advised that reading the Standard is heavy going and very time-consuming. But if you have more than a passing interest in the future of C, you should probably become an Observing member of the Committee just so that you receive copies of all working documents.

Identifiers

All internal identifiers are now significant in their first 31 characters. However, after a number of heated debates, the Committee agreed that it did not, and could not, control the name space acceptable to assemblers, librarians and linkers, etc., and voted to make a conforming implementation acceptable if it allowed externals of only one case and/or with as few significant characters as six. The character set supported by some linkers may also disallow underscores in externals or may ignore them; in which case, the following externals may all be identical:

```
proc
Proc
PROc
_proc
_Proc
pro_c
proc_
```

An internal identifier is a macro name or an identifier that does not have external linkage. External identifiers include external variables and functions.

It is strongly suggested that you do not define identifiers spelled with a leading underscore. This name space is "reserved" for implementers and future versions of the Standard. For example, new predefined identifiers are __DATE__ and __STDC__.

Structure and Union Name Space

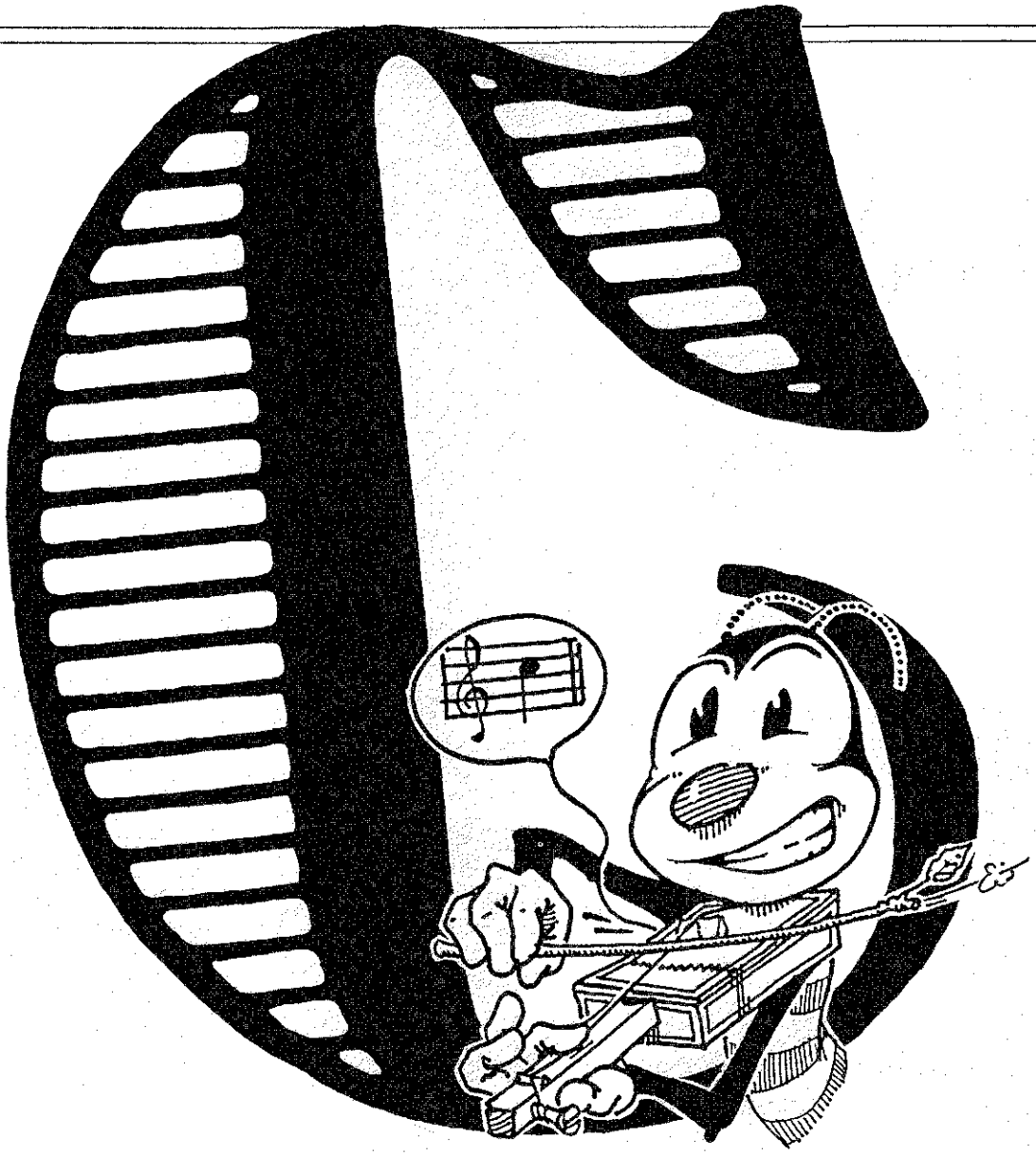
K&R (page 197) says, "Two structures may share a common initial sequence of members; that is, the same member may appear in two different structures if it has the same type in both and all previous members are the same in both. (Actually, the compiler checks only that a name in two different structures has the same type and offset in both, but if preceding members differ the construction is nonportable.)"

This meant that the following declarations generated a syntax error since i and l had different types and offsets.

```
struct s1 {
        int i;
        long l;
};

struct s2 {
        long i;
        int l;
};
```

This seems to have been an arbitrary restriction since member references are almost always qualified with a structure name

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



or pointer. The only exception could have been in that K&R allowed a structure (or union) member reference to begin with a constant as follows:

```
k = 0xff100->i;
```

where 0xff100 was an absolute memory address (*K&R*, page 209). Only in this case is the reference to i ambiguous—the compiler can't deduce i's offset and type. The Standard now requires that *each* structure and union have its *own* name space, so now you may use the same member name with different types and offsets as you wish without conflict. A side effect of this is that to use an absolute memory address "pointer" as shown above, you will first need to cast that value to a pointer to the corresponding structure (or union) as follows:

```
k = (struct s1 *)0xff100->i;
```

Now, member i is fully qualified, so its type and offset is known at compile-time.

Null Data and Function Pointers

The purpose of the macro NULL is to set or test for "the null data pointer." This is typically used with return values from library (especially I/O) functions. However, you may also use it to indicate the end or a special case in an array of data pointers. One particular, thorny problem is passing NULL to a function to indicate a null data pointer using something like

```
i = f(NULL);
```

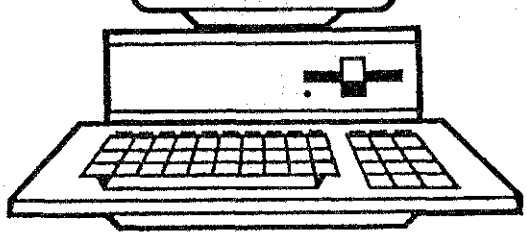
Generally, stdio.h contains some conditional compilation code to check the model size being used and then defines NULL as either 0 or 0L. This is necessary since data pointers can be either 16 or 32 bits in size. If function expects a 32-bit pointer, 0L *must* be passed. If it expects a 16-bit pointer, 0 must be passed.

It is *absolutely* necessary that you include stdio.h so that NULL is properly defined and that you *do not* explicitly define NULL erroneously yourself. If NULL is of the incorrect width value of zero, the argument list on the stack will be misinterpreted.

This whole model "nonsense" can be avoided by defining

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
comes to life on
Mnematics Videotex



Mnematics Videotex brings Programmer's Journal to LIFE!!
Download source code and other information from Programmer's Journal.
Coorespond electronically with the editors and authors.
MnEmail™ Electronic Mail, Realtime™ Chat, plus much more...
We'll waive the $39.95 sign up fee, if you call and subscribe now:
1-800-322-3633 (914-365-0184 in N.Y. State)
or write:
Mnematics, Inc., Sparkill, N.Y. 10976-0019

Number 65 on the Reader Service Card Mnematics Mailbox 6602

TURBO PASCAL GENERATOR

THE GTP PROFESSIONAL MODEL

Generate error-free Turbo Pascal source code for:

DATA BASES MENUS
SCREENS REPORTS

Build complete, working programs in minutes!

FEATURES
- Indexed Data Bases
- Multiple Screens
- Automatic Updating
- Built-in Edits
- Retrieval Facility
- Automatic File Build
- Quick Screen Handling
- Speedy DB Access
- Context Sensitive HELP
- Full Keyboard Support

EASY
1. Paint-the-screen
2. Define fields & calculations
3. Generate, compile & RUN

FLEXIBLE
- 100% Turbo Pascal
- Modify in Pascal under Turbo's Editor
- Extensions are easy with 130 page Programmer Reference manual
- 100 documented source code routines included

NOT Copy Protected NO Royalty Fees NO Run-time Library Required

Requires: IBM PC (100% compatible) 256K RAM 2 Disk Drives
PC-DOS 2.0+ Turbo Pascal 3.0 from Borland International

PRICE $200.00
CALL FOR QUANTITY DISCOUNTS
VISA/MC – Check – Money Order – No COD or Purchase Orders
Texas Residents: add $12.25 Sales Tax
Outside US & Canada: add $25.00 Air Postage and make payment by credit card or money order in US Funds

AEF
ALLEN, EMERSON & FRANKLIN
P.O. Box 928
Katy, TX 77492
(713) 391-8570

Number 53 on the Reader Service Card

NULL as follows:

```
#define NULL ((char *)0)
```

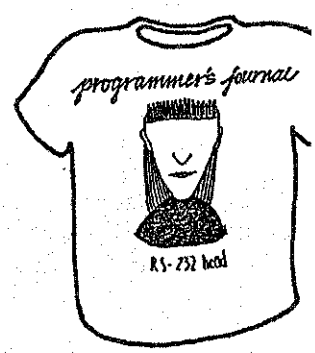
Now, NULL will always have the correct size regardless of the model being used. If you don't want to change NULL in stdio.h, you can invent a macro called NULLDPTR with the suggested value indicating the null data pointer value. The Standard has introduced a pointer to void and suggests that NULL be defined, using it as follows:

```
#define NULL ((void *)0)
```

The problem is more acute if you need a null function pointer. At least two of the commonly used memory models require data and function pointers to be of different sizes—in which case you *cannot* use NULL to indicate both a null data and a null function pointer in the same program since they will have different sizes. This is particularly so when using NULL

PARALLEL PORTS PJT102

These twin vessels are guaranteed to enhance communications with your favorite printer or anyone else you dump your text to. Multi-color imprint on a wine-colored T-Shirt.



RS-232 HEAD PJT103

Here's a connection you can't refi Plug your head into this neat-as-a-computer gray T-shirt with colorfi high-tech imprint.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

as function argument. Since data pointers are semantically quite different from function pointers, the two should never be mixed, anyway, and it is strongly suggested that you define a separate null function pointer as follows:

```
#define NULLFPTR ((void)(*)0)
```

Here, the null function pointer is zero cast to a pointer to a function that has a void return type. If your compiler doesn't support the type void, use int instead.

Future Meetings

The schedule for future ANSI meetings is

March 9-13, Boulder,Colorado
June 8-12, Paris, France
September 14-18, Boston, Massachusetts
December 7-11, Phoenix, Arizona □

Rex Jaeschke is an independent computer consultant, writer, and lecturer. He is also a member of the X3J11 C Standard's Committee and co-founder and Editor of The C Journal.

If you would like to comment on or question any aspect of the Proposed ANSI C Standard, write to

Rex Jaeschke
2051 Swans Neck Way
Reston, VA 22091

Rex will address your concerns in future columns as space permits.

COMMON C BUG PJT104

Everyone's got 'em! Now proudly display your own specimen. The bug is brown (naturally), crawling across a gold nameplate on a beige T-shirt.

Famous Cartoons Reformatted as T-Shirts

See inside back cover for details.

S P E E D W A R E™
A DIVISION OF AMERICAN AFFORDABLE SOFTWARE CORPORATION

PRESENTS

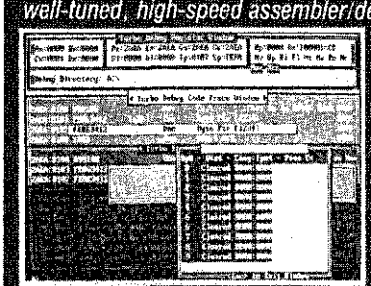
Turbo Debugger™ Features

- User-definable custom windows — Change window sizes or colors
- Single step tracing with "Back-Tracing" up to 100 instructions
- In-Line relocating Assembler with non-destructive insert capability
- Fully Symbolic, Create Comments and Line Labels, Keyboard macros
- Dual Monitor support, with Screensave
- Specify several Breakpoint conditions with the built-in Assembler
- Use NMI Breakout switch or User-definable hotkeys
- Read, Write and Modify .EXE files

Turbo Editasm™ Features

- Integrated WordStar compatible Editor
- Integrated MicroSoft compatible Macro Assembler
- Full 8088, 8086, 8087, 80186, 80287, 80286(P), & (V) support
- Full Macro expansion capability
- Built-in Calculator & Xreference Capability
- Limited DOS command support
- Supports In-Memory Assembly

To maximize your programming potential order the ultimate "Turbo Team" today — the first well-tuned, high-speed assembler/debugger for the IBM P.C. and compatibles family.



$99.00 Turbo Editasm™
$89.00 Turbo Debugger™

THE TURBO TEAM™
Inside California (800) 334-3664 • Outside California (800) 443-6190

$99.00 ☐ Turbo Editasm™
$89.00 ☐ Turbo Debugger™

Name _____
Shipping Address _____
City _____
State _____ Zip _____
Telephone _____
Amount Enclosed _____
Payment VISA MC BANK DRAFT CHECK
Credit Card Exp. Date _____
Card # _____

CA Residents add 6% sales tax. Outside USA add 10%, and make payment by bank draft payable in U.S. dollars drawn on a U.S. bank.

Dealer and Education Discounts Available.

Speedware™
9719 Lincoln Village Dr., Ste. 303
Sacramento, CA 95827

Number 72 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

More Tips on Formatting BASIC Data

by
Bernard H. Robinson, Jr.

The material presented here follows up my two previous articles, "Structuring BASIC Output with FIELD Statements" (*PJ*, Vol. 4, No.2) and "Tips on Formatting with BASIC PRINT USING Statements" (*PJ*, Vol. 4, No. 4). The programs that accompany the article have been tested on both IBM PC interpreted BASIC 2.00 and 3.2 on an IBM PC. The programs as presented will also compile on Microsoft QuickBASIC, Version 2, hereafter called "QB2." If you want to use the information presented, you are urged to verify for yourself that it is suitable for your use and works as stated on your system.

Loading Numeric Data in Binary Form into a Field Buffer

"Structuring BASIC Output with FIELD Statements" discussed the use of random access file buffers in structuring ASCII data for printing to screen or printer. These buffers are set up by executing an OPEN statement and structured by executing FIELD statements. Here we will explore the more conventional use of these buffers: the loading of a mixture of data types into them for transfer to a random access file. The data types include numeric data in binary form and ASCII data in strings. Numeric data is loaded in binary form to save file space—only two bytes (character spaces) are required to hold an integer, four bytes to hold a single precision number, and eight bytes to hold a double precision number.

To process numeric data for storing in the buffer in binary form, BASIC requires the use of the MKx$( ) family of functions, where x can be I, S, or D for converting integer, single precision, or double precision numbers, respectively. These are peculiar functions—they convert numeric values into what is nominally string form, but the contents of the string are the binary form of the number, not ASCII characters. The bytes in the string will be identical to the bytes that represent the number as a numeric variable or constant.

When transferring data out of a buffer, BASIC requires that data that was processed with MKx$( ) functions be reconverted with corresponding CVx( ) functions before it can be used as numeric data. The system of using MKx$( ) and CVx( ) functions simplifies the portion of a BASIC interpreter or compiler that handles random access files. These functions must never be used in connection with sequential access files. Sequential access files can handle only data in true ASCII strings.

Don't be mislead by the fact that the result of an MKx$( ) function is described as a string. It is a string only in the manner in which BASIC stores it internally before inserting it into the buffer. That is, BASIC stores numeric variables and string variables internally in different formats. The MKx$( ) functions change the format in which the numeric data is stored to that of a string variable, but the data part of the string variable contains the original binary data instead of the ASCII equivalent of the numeric data that would have been generated by a STR$( ) function. Since the code for the BASIC interpreter is written for storing string variables into the buffer and retrieving string variables from the buffer, the use of MKx$( ) and CVx( ) functions allows much of the same code to be used to store and retrieve numeric variables in binary form.

If you want more details, a description of how interpreted BASIC stores numeric and string variables may be found in the BASIC manual. In the manual I use, it is found in Appendix I, pages 3 and 4. QB2 uses a different format for string variables; see pages 143 and 205 in the QB2 manual.

The program PROG1.BAS (Figure 1) is a tutorial device designed to help you understand the loading of numeric data in binary form into a buffer and to demonstrate what the MKI$( ) function really does. In this program, the user is asked to input an integer number, the number is checked to be sure it is in the right range, and the number is then converted with the MKI$( ) function and set into a file buffer. The data in the file buffer is then extracted and checked byte by byte. Finally the data from the buffer is used to construct the binary equivalent of the number as a printable ASCII string in the buffer.

Line 1070 opens a random access file to establish a buffer. Line 1050 establishes a field 2 bytes long named N$, which will hold the integer in binary form. Line 1050 also establishes a 16-byte field, BIN$, which will hold the ASCII string equivalent of the binary value of the number.

Line 1090 defines NL$ as the first byte and NH$ as the second byte of the N$ field. The L(ow) and H(igh) parts of these names emphasize that BASIC stores the bytes of a word (integer) in what might appear to be reverse order: The first byte contains the lower order byte of the integer, and the second byte contains the higher order byte. Actually, this follows logically from the fact that each byte in memory is addressable individually by the 8088 microprocessor, with the lower order byte having the lower address. BASIC just puts the bytes into the buffer in the numerical order of their addresses in memory. Line 1090 also

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

defines four 4-byte fields that will be used in building the BIN$ field of line 1080. Line 1100 defines an 18-byte field that includes all the fields of lines 1080 and 1090.

Line 1110 converts the input integer N1% with the MKI$( ) function and puts it into the 2-byte buffer field N$. We used the LSET function to put the data into the buffer, but we could just as well have used the RSET function since the data is the same length (2 bytes) as the field in the buffer. At this point we would like to know the numerical values stored in each of the one-byte strings NL$ and NH$. The VAL( ) function is usually used to convert strings to numerical values, but it is not the correct function to use here. The VAL( ) function works correctly only if the string contains the number in a form in which a printable ASCII character represents each decimal digit of the number. NH$ and NL$ are not in this form; rather, each contains a single byte of binary data that represents a value from 0 to 255 inclusive.

The ASC( ) function is the correct one to return the numerical value of these single bytes of binary data. Line 1120 uses the ASC( ) function to do just this, and line 1130 prints the values of the two bytes.

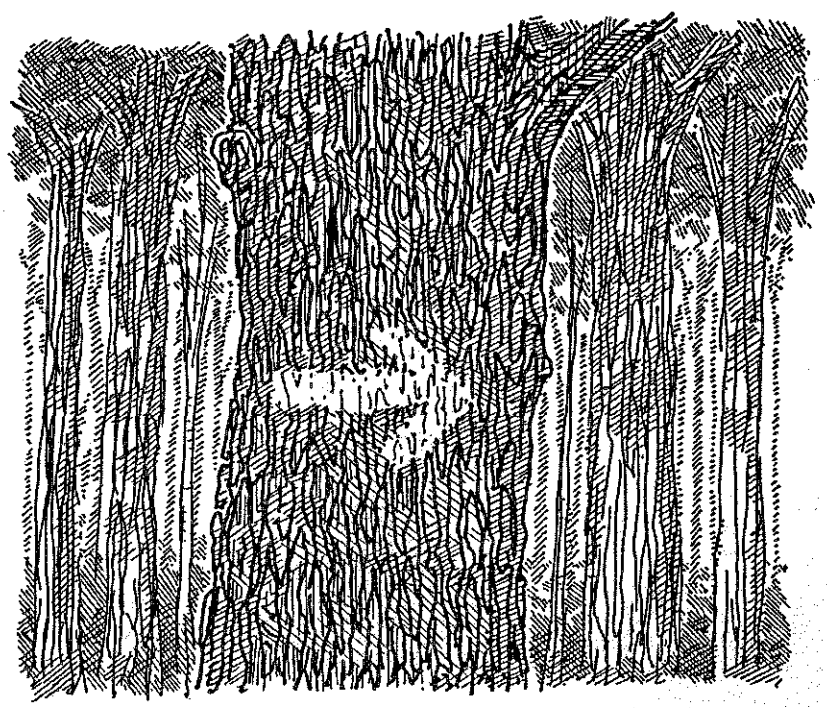
We now want to check whether these two values are the expected values. If the original number is positive, 256 times the high byte value plus the value in the low byte should equal the original number. However, if the value is negative, 65536 must be subtracted from the results of the computation. If the input number is negative, the high byte is equal to or greater than 128; this byte is tested in line 1140 to determine which computation is to be made. One or the other computation is made (line 1150 or 1160), and the resulting number is printed. If this number is indeed equal to the input number for all values of input, this verifies that our interpretation of the way MKI$( ) operates and of the way binary data is stored in the field buffer is correct.

As a further exercise, we wish to construct a 16-byte printable ASCII string containing the binary equivalent of the input integer. One approach is to convert the input integer into its 4 hexadecimal digits and then use the well known bit pattern for each of these digits. We could use the HEX$( ) function on the original input number to obtain the 4 hex digits; however, to further confirm our understanding of what we did above, let's operate on NH% and NL%, the high and low byte values of the input number. Each of these bytes can be converted into two hex digits. If the byte value is zero to 15 decimal, the HEX$( ) function will return only one digit, so in these cases we need to prepend an ASCII 0 to obtain two digits in the results (lines 1170 and 1180). The 16 possible bit patterns corresponding to a single hex digit were stored into the array B$( ) in the subroutine at line 2000. Lines 1190 through 1220 and the subroutine at line 3000 determine the index into B$( ) for the bit pattern corresponding to each hex digit of the input number. As each binary bit pattern is determined, it is LSET into the field buffer in its proper place. Line 1230 prints the binary equivalent. The reason for the Z$ field is discussed in the next section.

The File Buffer as a Data Structure Definer/Interpreter

Some programming languages, such as C and assembly, have what are called data structures. These are groups of data variables that may be mixtures of various data types: integers, single precision numbers, double precision numbers, arrays, strings, or even other structures. Structures are designed to be referenced as a group by a single name and as individual variables by the structure name plus additional field identifiers. For example, a structure named CUSTOMER might contain an ASCII field named NAME that contains the customer's name. Other data fields of various data types that pertain to a customer could make up the rest of the structure. Then an array of CUSTOMER structures can be defined so that CUSTOMER(0) would hold the data structure for one customer, CUSTOMER(1) that for the next, etc. The way to refer to an individual field for a particular structure is dependent on the language, but might be something like CUSTOMER.NAME(1), for example.

While BASIC has no direct equivalent of these data structures, the buffer set up by the FIELD statements of a random access file can help to perform some of the same functions that are performed by using



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

structures in other languages. In line 1100 of PROG1.BAS, the field variable Z$ is defined as the first 18 bytes of the buffer. It is printed as the last line of the output of the program. The first two characters of Z$ are the 2 bytes of the binary number n1%, low byte first. They will be printed as ASCII characters corresponding to the binary data in the 2 bytes and will look nothing like the numeric data they represent. If one of these bytes has the value 12, the display screen will respond as though a CLS statement had been executed. If the value 7 is present, the speaker will BEEP. If the value is above 128 or below 32, some weird character is apt to be displayed. The last 16 bytes of Z$ make up the ASCII string equivalent of the binary number and will print normally. Of course, the binary data is not intended for display, and we are printing it to the screen just to show that there is something there in the Z$ string. This is a trivial example of a binary field and an ASCII field in the same file buffer.

A random access file buffer can be defined to match the desired fields of a structure. Data can be placed in the individual fields using the LSET or RSET functions. Any numerical data should be converted with the appropriate MK$( ) function and placed in fields of corresponding length. Then the entire structure (buffer contents) can be stored into single string with a simple assignment statement such as

```
1000 CUSTOMER$(0)=Z$
```

where Z$ is a field defined to include the whole contents of the buffer. For interpreted BASIC, the LSET statement would be preferred over a simple assignment statement in order to minimize the generation of string garbage. To reference the individual fields of a structure stored in a string, the string can be placed back in the buffer with a statement such as

```
2000 LSET Z$=CUSTOMER$(0)
```

and the individual fields can be accessed through the defined fields of the buffer.

For interpreted BASIC, the maximum string length allowed is 255 bytes, so the total number of characters in one such "structure" could not exceed 255. However, QB2 allows strings to be up to 32,767 bytes long.

These techniques are especially likely to be useful if structured data is to be stored in a disk file or retrieved from a disk file and we also want to select certain of the structures to be retained in memory for sorting or some other operation. As an example of a possible use of this technique, consider that QB2, which comes with INT86 and INT86X routines for executing DOS interrupts, gives the BASIC programmer relatively easy access to disk directory information. The data returned by a call to function 78 (hex 4E—find first matching file) or to function 79 (hex 4F—find next matching file) is 43 bytes of structured information. These 43 bytes of returned data for each file could be stored in an element of a string array as they are returned. Later each could be set into a suitably defined buffer for access to the individual data fields.

Alternate Methods

Of course the use of a random file buffer is not the only way in BASIC to break a string into substrings or construct strings from substrings. The LEFT$( ), RIGHT$( ), and MID$( ) functions can isolate particular sections of a string. The MID$( ) function in particular is very powerful. The MID$( )=string statement is also a powerful way to modify or construct strings. The concatenation operator + and the SPACE$( ) function also aid in building strings. Almost anything in the way of formatting that can be done with a file buffer could be done with these functions and statements. Which approach is best in a given instance depends upon the particular circumstances and on personal preference. If data is to be output to or input from a random access file, of course, a file buffer is necessary.

PROG2.BAS (Figure 2) was written to perform the same functions as PROG1.BAS without using a file buffer. The main formatting functions used were MID$( ), LEFT$( ), RIGHT$( ), LSET, and RSET. Here the purpose is to show that the binary data produced by a MKI$( ) function can be put into an ordinary string variable with a LSET statement and that data can be extracted and evaluated in a way very similar to that used in PROG1.BAS. Since this program is so similar to PROG1.BAS, little discussion seems necessary. Lines 1060 through 1110 establish the length of several strings. This is necessary for strings in which data will be placed with LSET, RSET, or the MID$( ) functions. In order to reduce the generation of string garbage in interpreted BASIC, an LSET or RSET function is sometimes used where a simple assignment could have been used.

Data structures can also be simulated without using a file buffer by extensive use of the functions and statements discussed above. Writing subroutines for packing the desired data fields into a single string variable and for unpacking the data fields from a single string variable would be a logical first step in using this approach in a program.

Performance of PROG1.BAS and PROG2.BAS on Interpreted BASIC

PROG1.BAS and PROG2.BAS were first developed with interpreted BASIC. So that the approaches taken in the two programs could be evaluated, special test versions of the programs were derived from the listed versions. The test program for PROG1.BAS added a loop with 1000 iterations that enclosed from line 1110 through line 1220 and used a fixed numerical input of 25,000. The PRINT statements at lines 1130, 1150, and 1160 were commented out. The TIMER function was used to record the starting time just before the first loop and the ending time just after the last loop to determine the time required to complete 1000 loops. PROG2.BAS was modified in a very similar way for testing. On an IBM PC using interpreted BASIC 3.2, PROG1.BAS required 123.5 seconds and PROG2.BAS required 148.0 seconds for 1000 loops through the program.

Besides execution time, the amount of string garbage produced by the programs is a consideration because these approaches may be taken in programs that process a lot of data. Statements to print the free string space were added before the first loop and after the last loop so that the decrease in string space due to string garbage could be determined. Neither program generated more than 100 bytes total string garbage for the entire 1000 loops through the program.

Compiling PROG1.BAS and PROG2.BAS with QB2

The programs were also compiled and run on QB2. Since we wanted the published programs to be able to run on either the interpreter or the compiler, the programs do not take advantage of such QB2 features as named subroutines, separate compilation of subroutines, no line numbers, etc.

A couple of changes from my first interpreted versions were necessary in order to compile the programs. The first problem was that QB2 is fussier about integer overflow. The computation on line 1160 was originally

```
NH%*256+NL%-65536!
```

which worked fine interpreted, but gave an "Overflow" error on QB2. Converting NH% to a single precision equivalent, N2, and using that variable in place of NH%

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

gave correct results either compiled or interpreted.

The second problem was that with interpreted BASIC, the dummy file opened to establish the file buffer could be closed after the execution of the FIELD statements without disturbing the buffer as long as the file number was not used in another OPEN statement, but with QB2, the buffer ceases to exist when the file is closed. This was handled by keeping the file open until the program is finished with the buffer.

When the printed versions of the programs are compiled, they run more than ten times faster than when interpreted, but the techniques which reduced string garbage generation for the interpreter did not work as well for QB2. For 1000 iterations, PROG1.BAS consumed about 24000 bytes in string garbage and PROG2.BAS consumed about 36000 bytes. When the statement

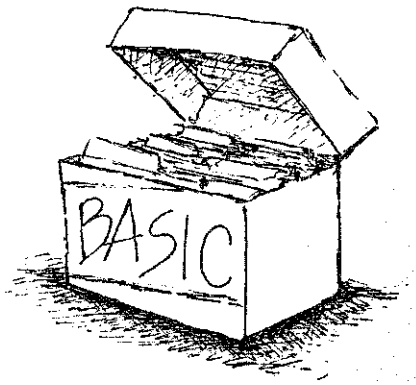
```
XFRE=FRE("")
```

was inserted in the loop to force garbage collection at the beginning of each pass through the program loop, the total garbage generated in 1000 loops dropped to less than 100 bytes for the compiled versions. Before the statement to generate garbage collection was added, PROG1.BAS ran 1000 iterations in about 8.4 seconds; adding the statement increased the time to about 13.5 seconds for 1000 iterations. For PROG2.BAS, the "before" runtime was about 11.4 seconds and the "after" was about 16.4 seconds for 1000 iterations. The compiled programs after this change ran approximately ten times faster than the interpreted programs. A time of about 5 milliseconds to collect the small amount of string garbage generated during each pass through the program seems quite efficient to me.

The times were obtained by compiling without the debug option although the debug option was used initially in debugging the programs. Similar times were obtained whether the programs were run from the QB2 menu (compiling in memory) or from .EXE versions of the programs.

Although I have used it only a little so far, my first impressions of QB2 are very positive. It is generally compatible with the IBM PC BASIC interpreter and also with the Microsoft GW-BASIC used on many compatibles. If you can program in these languages, you can start using QB2 with minimum learning time. The differences from interpreted BASIC seem logical and give QB2 added capabilities. The development environment provides a program editor and menu screens for easy compiling and linking of programs. I could complain that I find some of its editor commands awkward, but the tie-in of the editor to the error reporting of the compiler is often a big help in debugging programs. The manual seems obscure in some places, but it is about 600 pages long and contains a wealth of information. All in all, I feel that QB2 is a great value for its price. □

Code follows.



PolyMake *The Leading Make Utility*

Are you still using a prehistoric Make? Now, step up to PolyMake, the most powerful and flexible Make utility available for programmers using MS-DOS. PolyMake is like an intelligent assistant that remembers how to rebuild a program when you change one part of the program. PolyMake will automatically invoke your compiler, assembler, linker, librarian or other tools to update a single program or entire software systems when you type — MAKE. PolyMake comes with built-in rules for rebuilding programs but you can also teach it new rules so you don't have to remember the file dependencies in your program. Advanced programmers prefer capabilities like fully recursive makefile processing and the ability to invoke PolyMake with special "flags" and macros that automate a whole range tasks. New Make users appreciate the Step-By-Step tutorial and intuitive commands. Handles source files written in any language. Requires DOS 2.0 & higher. Compatible with LANs, the IBM PC, XT, AT and other MS-DOS PCs. For complete details write for the POLYTRON Programmer's Catalog. $99

PVCS *The Most Powerful & Flexible Source Code Revision & Version Control System.*

The POLYTRON Version Control System (PVCS) allows programmers, project managers, librarians and system administrators to effectively control the proliferation of revisions and versions of source code in software systems and products. PVCS is a superb tool for programmers and programming teams. (A special LAN version is also available.) If you allow simultaneous changes to a module PVCS can merge the changes into a single new revision. If changes conflict, the user is notified. Powerful capabilities include: Stores and retrieves multiple revisions of text; Maintains a complete history of revisions to act as an "audit trail" to monitor the evolution of a software system; Maintains separate lines of development or "branching"; Provides for levels of security to assure system integrity; Uses an intelligent "difference detection" to minimize the amount of disk space required to store a new version. Requires DOS 2.0 or higher. Compatible with the IBM PC, XT, AT and other MS-DOS PCs. Single User version $395. 5-station LAN version $1,000, add $500 for each additional 5 stations. $395

TO ORDER: VISA/MC 1-800-547-4000, Dept. No.306; Oregon/Outside US, 503-684-3000
Send Checks/POs To: POLYTRON Corp. 1815 NW 169th Pl., #2110, Dept. No. 304, Beaverton, OR 97006

POLYTRON
High Quality Software Since 1982 ®

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1.

```
10 ' Program name: PROG1.BAS
20 '      Purpose: Demonstration of use of FIELD statements
30 '   Written by: D. H. Robinson, Jr.
40 '         Date: October 26, 1986
50 '      Version: 018
1000 DIM B$(15)
1010 GOSUB 2000 ' Load array
1020 PRINT "Enter an integer number in the range of -32768 to +32767, "
1030 LINE INPUT "or enter Q to end the program: ";N1$
1040 QX%=INSTR(N1$,"Q") + INSTR(N1$,"q") : IF QX%<>0 THEN END
1050 NX=VAL(N1$) : IF NX <-32768! OR NX >32767! THEN BEEP : GOTO 1020
1060 N1%=NX : HH$=SPACE$(2) : HL$=SPACE$(2)
1070 OPEN "QQQ.WW" AS #1 LEN=32
1080 FIELD #1, 2 AS N$,16 AS BIN$
1090 FIELD #1, 1 AS NL$, 1 AS NH$,4 AS H4$,4 AS H3$,4 AS H2$,4 AS H1$
1100 FIELD #1,18 AS Z$
1110 LSET N$=MKI$(N1%)
1120 NH%=ASC(NH$) : NL%=ASC(NL$)
1130 PRINT "NH% is: ";NH%;" and NL% is: ";NL%
1140 N2 = NH% : IF NH%>=128 THEN GOTO 1160
1150 PRINT "NH%*256+NL% = ";NH%*256+NL% : GOTO 1170
1160 PRINT "NH%*256+nl%-65356 = ";(N2*256)+NL%-65536!
1170 IF NH%>=16 THEN LSET HH$=HEX$(NH%) ELSE LSET HH$="0"+HEX$(NH%)
1180 IF NL%>=16 THEN LSET HL$=HEX$(NL%) ELSE LSET HL$="0"+HEX$(NL%)
1190 Q=ASC(LEFT$(HH$,1)) : GOSUB 3000 : LSET H4$=B$(X)
1200 Q=ASC(RIGHT$(HH$,1)) : GOSUB 3000 : LSET H3$=B$(X)
1210 Q=ASC(LEFT$(HL$,1)) : GOSUB 3000 : LSET H2$=B$(X)
1220 Q=ASC(RIGHT$(HL$,1)) : GOSUB 3000 : LSET H1$=B$(X)
1230 PRINT "Binary value of ";N1%;" is: ";BIN$
1240 PRINT "Z$ is: ";Z$
1250 PRINT : PRINT
1260 CLOSE #1
1270 GOTO 1020
1280 '
2000 ' Load array
2010 DATA "0000","0001","0010","0011","0100","0101","0110","0111"
2020 DATA "1000","1001","1010","1011","1100","1101","1110","1111"
2030 FOR I=0 TO 15
2040 READ B$(I) : NEXT I
2050 RETURN
2060 '
3000 ' Compute index into B$() for a hex character
3010 IF Q>=48 AND Q<=57 THEN X=Q-48 : GOTO 3030
3020 IF Q>=65 AND Q<=70 THEN X=Q-55
3030 RETURN
```

FIGURE 2.

```
10 ' Program name: PROG2.BAS
20 '      Purpose: Demonstration of use of MID$() statements
30 '   Written by: D. H. Robinson, Jr.
40 '         Date: October 26, 1986
50 '      Version: 018
1000 DIM B$(15)
1010 GOSUB 2000 ' Load array
1020 PRINT "Enter an integer number in the range of -32768 to +32767,
1030 LINE INPUT "or enter Q to end the program: ";N1$
1040 QX%=INSTR(N1$,"Q") + INSTR(N1$,"q") : IF QX%<>0 THEN END
1050 NX=VAL(N1$) : IF NX <-32768! OR NX >32767! THEN BEEP : GOTO 1020
1060 N1%=NX : HH$=SPACE$(2) : HL$=SPACE$(2)
1070 N$=SPACE$(2) : BIN$=SPACE$(16)
1080 NL$=SPACE$(1) : NH$=SPACE$(1)
1090 H4$=SPACE$(4) : H3$=SPACE$(4)
1100 H2$=SPACE$(4) : H1$=SPACE$(4)
1110 Z$=SPACE$(18)
1120 LSET N$=MKI$(N1%)
1130 LSET NL$=LEFT$(N$,1) : LSET NH$=RIGHT$(N$,1)
1140 NH%=ASC(NH$) : NL%=ASC(NL$)
1150 PRINT "NH% is: ";NH%;" and NL% is: ";NL%
1160 N2=NH% : IF NH%>=128 THEN GOTO 1180
1170 PRINT "NH%*256+NL% = ";NH%*256+NL% : GOTO 1190
1180 PRINT "NH%*256+nl%-65356 = ";(N2*256)+NL%-65536!
1190 IF NH%>=16 THEN LSET HH$=HEX$(NH%) ELSE LSET HH$="0"+HEX$(NH%)
1200 IF NL%>=16 THEN LSET HL$=HEX$(NL%) ELSE LSET HL$="0"+HEX$(NL%)
1210 Q=ASC(LEFT$(HH$,1)) : GOSUB 3000 : LSET H4$=B$(X)
1220 Q=ASC(RIGHT$(HH$,1)) : GOSUB 3000 : LSET H3$=B$(X)
1230 Q=ASC(LEFT$(HL$,1)) : GOSUB 3000 : LSET H2$=B$(X)
1240 Q=ASC(RIGHT$(HL$,1)) : GOSUB 3000 : LSET H1$=B$(X)
1250 MID$(BIN$,1,4)=H4$ : MID$(BIN$,5,4)=H3$
1260 MID$(BIN$,9,4)=H2$ : MID$(BIN$,13,4)=H1$
1270 PRINT "Binary value of ";N1%;" is: ";BIN$
1280 MID$(Z$,1,1)=MID$(N$,1,1) : MID$(Z$,2,1)=MID$(N$,2,1)
1290 MID$(Z$,3,16)=BIN$
1300 PRINT "Z$ is: ";Z$
1310 PRINT : PRINT
1320 GOTO 1020
1330 '
2000 ' Load array.
2010 DATA "0000","0001","0010","0011","0100","0101","0110","0111"
2020 DATA "1000","1001","1010","1011","1100","1101","1110","1111"
2030 FOR I=0 TO 15
2040 READ B$(I) : NEXT I
2050 RETURN
2060 '
3000 ' Compute index into B$() for a hex character
3010 IF Q>=48 AND Q<=57 THEN X=Q-48 : GOTO 3030
3020 IF Q>=65 AND Q<=70 THEN X=Q-55
3030 RETURN
```


C CODE FOR THE PC

source code, of course

GraphiC 3.0 hi-res color plots	$300
Panache C Program Generator	$125
QC88 C Compiler	$90
Concurrent C	$45
Coder's Prolog in C	$45
Biggerstaff's System Tools	$40
Translate Rules to C	$30
LEX	$25
YACC & PREP	$25
tiny-c interpreter & shell	$20
C Tools	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
(512) 258-0785

Free shipping on prepaid orders **No credit cards**

Number 30 on the Reader Service Card

Sort RM/COBOL Files

• • • QISORTC • • •

Finally, a sort utility for RM/COBOL.
Using efficient algorithms in assembly language making it the fastest sort on the MS-DOS market. Qisortc supports:

- RM/COBOL Version 1.5x & 2.xx files
  - Single / Dual File Indexed
  - Relative and Sequential
- ASCII, COMP-1, & COMP-3 fields
- 32 sort control fields
  - ascending / descending
- 32 record selection fields
  - EQ:NE:GT:GE:LT:LE field:constant
  - implied AND, OR may be specified
- 32 output fields or total record
- CALL from RM/COBOL, when provided subroutine is linked with runtime
- Use pathnames and return codes
- Keyboard or Parameter file
- Requires MS-DOS ver 2.xx or above
- Only $95.00 - COD, VISA, or M/C.

Quantumn Information Systems, Inc.
145 N. W. 85th St., Suite 103
Seattle, Washington,98117
(206) 789-2888 Telex 4941041

Number 31 on the Reader Service Ca

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Turbo Extender and T-DebugPLUS
TurboPower Software
478 W. Hamilton Ave., Suite 196
Campbell, CA 95008

Turbo Extender
A set of Turbo Pascal utilities, the most notable of which guides you through the conversion of an overlayed or chained Turbo program into either a "large code" form that consists of modules that can be dynamically loaded and linked at runtime or into a true .EXE file.

You organize your code into modules of up to 64K each and then compile them using Turbo's chain option. This means that each module must be capable of compiling alone. If code in a module calls an external procedure, the utilities generate a shell procedure within the module to placate the compiler. The shell also serves as the jumping-off place to the external procedure when the modules are eventually linked. Turbo Extender also generates a final begin and end statement for the module if needed and gathers the program's global data declarations for the module.

Each module is given its own copy of the 10K Turbo library.

When the modules are finally linked, long calls happen only between modules. Procedures within a module all use near calls.

The resulting program uses only the single original Turbo data segment and a single heap.

Turbo Extender diverges from Turbo syntax in an annoying way—the word "overlay" needs to be on the same line as the word "procedure." The large source file we used for a test always had the word "overlay" on a separate line.

One concern when converting to the .EXE model is that variables that are declared in the code instead of data segment (Turbo's *typed constants*) are duplicated in each module. If one of the modules alters a code variable, the other modules won't know about it. TurboPower provides a procedure that updates all the other modules.

Other utilities in the TurboPower package include BIGARRAY, which supports five different models of arrays larger then 64K; a runtime disk cache; a Pascal source code encryptor (in case you want to distribute encrypted library routines); and OVERMAP, an overlay mapper.

T-DebugPLUS
With T-DebugPLUS, you can view your source code while debugging; trace program operation statement by statement; insert breakpoints by procedure, function name, or statement number; and examine and change global or local variables including strings, array variables, records, and sets.

The program has separate screens for debugging and program output. You can view the actual program output with a keystroke. The program will debug programs in the compiler's memory mode only—no debugging of .COM programs is supported. A program may contain overlays, but the overlays themselves can't be debugged.

C-Index+
Trio Systems
2210 Wilshire Blvd., Suite 289
Santa Monica, CA 90403
(213) 394-0796
A library of B-tree disk data management utilities for C language programmers. The C-Index is designed for single-user and multi-user applications and includes machine-independent record locking. It supports Lattice and Microsoft 3.0, C86 with precompiled libraries, and Xenix. For moving to other compilers and environments, complete source code is included, along with instructions and test programs to validate the port.

A data structure can be defined within your program, even while running. The address of the structure array is then passed to C-Index, which builds a compressed, variable length record based on the "datalist" description.

UR/FORTH 1.0
Application development system $350
Object module libraries $500
Source code license $1500

Laboratory Microsystems
3007 Washington Blvd, Ste. 230
Marina Del Rey, CA 90292 5548
A direct-threaded-code Forth interpreter/compiler that holds the top of the stack in a register. A *Forth-83 Standard* implementation.

UR/FORTH features a segmented memory model, with code, data, headers, and dictionary hash table in separate segments. Separation of code and data means UR/FORTH can run in virtual addressing, memory-protected systems such as 68020 or 80286 Unix and Xenix.

UR/FORTH modules may be linked with other language object modules. Also featured are "well-behaved" dynamic memory allocation, relocatable binary overlays, a completely hashed dictionary for fast compilation, string handling operators, and a dynamic string storage manager.

Headerless turnkey applications may be distributed without further royalties or fees.

The threaded code implementation compiles about twice as fast and runs about 25% faster than PC/FORTH.

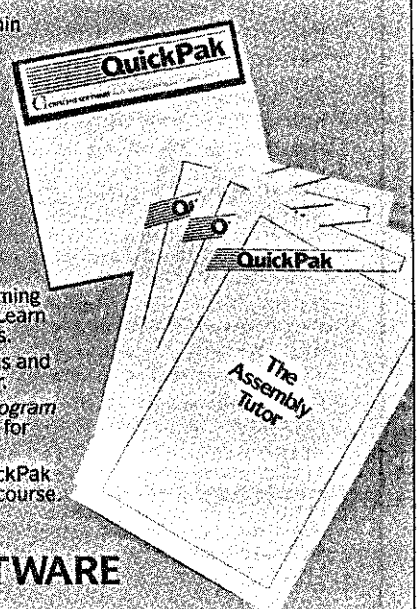
QuickBASIC just got quicker with
QuickPak

QuickPak is a superb collection of enhancements, subroutines, and instructional material designed to help you get the most out of programming in BASIC.

- Powerful assembly language routines to give your programs more speed, more power, and full access to DOS and BIOS services. SORT all or part of a string array with one command! FIND any string or sub-string within an entire array *regardless of capitalization* — accepts wildcards. READ directories into your programs from any drive or path. READ/WRITE disk sectors — create your own DOS utilities! CLEAR or SCROLL any part of the screen. MANY, many more programs included.
- Professionally written QuickBASIC routines and functions. Powerful input routines for text, dates, and numbers. Menus, scroll bars, date/time functions, and much more.
- The Assembly Tutor — a complete guide to learning assembly language *from a BASIC perspective*. Learn how to create your own routines and extensions.
- Tips and Tricks book — packed with clever ideas and techniques to help *you* be a better programmer.

You get all this, all of the source code for *every program included*, and a thirty-day money back guarantee for only $69.00.

No royalties are required for using any of the QuickPak routines in your programs. Not copy protected, of course.



by
CRESCENT SOFTWARE
64 Fort Point Street
East Norwalk, CT 06855
(203) 846-2500

QuickPak requires Microsoft QuickBASIC or BASCOM, DOS 2.0 or higher. Visa, M/C, C.O.D., or checks accepted.

Number 83 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PopScreen
Baysoft
Box 6562
Albany, CA 94706
(415) 527-6894
A screen generator that doesn't use resident loaders or externally maintained screen files. All screen materials are placed in a linkable object file, so they can be integrated into the programmer's code. The result is no disk access and fast screen popups.

PopScreen is designed to work with Assembler; Lattice and Microsoft C; and IBM, Microsoft and Turbo Pascal.

Lightning WordWizard Toolbox
Turbo Lightning
Borland International
Lightening WordWizard was on display at the Borland booth at the recent San Francisco PC Faire. It's a slick compendium of the kinds of word games my mother and grandfather did at the kitchen table for twenty years. It was written by Borland to demonstrate Turbo Lightning, their RAM resident word look-up engine based on the Random House Dictionary and Thesarus. Borland is marketing the source code, along with a 182-page programmer's technical reference for writing your own applications using the engine, as the Lightning WordWizard Toolbox.

The source code is a prize in itself: clear, consistent Pascal well worth studying. You can include parts of the code in your own applications, as long as you don't just try to market the whole thing.

The first release of the engine supports these 16 calls: check word spelling, return synonyms, return sound-alike words, return word number for word, get a word from its word number, switch dictionaries, change environment, return list of words in given length range.

The engine is called by generating an interrupt 16h (keyboard interrupt) with the appropriate registers loaded.

The engine uses two different numbering systems to keep track of dictionary words. The global number numbers all the words in the dictionary from beginning to end. The block and intra-block word number (also called the offset in the block) identifies the actual physical disk block to read and the word position within the block.

Words, especially the first four letters of words, are usually stored in alphabetical order. Word suffixes are separately encoded.

QuickPak
$69.00
Crescent Software
64 Fort Point Street
East Norwalk, CT 06855
A set of utilities for users of QuickBASIC and Version 2 of the IBM/Microsoft BASCOM compiler. Included are three manuals, four input routines, three menu programs, 19 assembly language routines, and more than a dozen BASIC functions. Source code is included.

To install QuickPak with QuickBASIC: QuickBASIC is called with QB /L QuickPak.Exe. Then the package's routines are accessed with QuickBASIC's CALL command.

Command Plus
ESP Software Systems Inc.
11965 Venice Blvd., Suite 309
LA, CA 90066
(213) 390-7408
A command processor for DOS that replaces COMMAND.COM in Versions 2.0 and above. Command Plus includes all the features of COMMAND.COM, but adds a script processor, command line editing, pattern matching in file specifications, redirection of standard I/O, piping, command recall, command line macros, a user-specified environment size, a directory stack, a command log file, and other extensions. Standard commands, such as DIR, COPY, and DEL are enhanced. Also included is BROWSE, a quick file viewer. Looks like a powerful and interesting package.

Fast88
MicroSpeed, Inc.
5307 Randall Place
Fremont, CA 94538
(415) 490-1403
The Fast88 is a low-cost (compared to 286 accelerators) 8088 speed-up option that replaces the 8088 CPU with a NEC V-20 and ups the clock rate to 6.1 MHz, 6.7 MHz, or 7.4 MHz. The system comes with an internal board with a jumper-selectable clock and an external control with a high/low speed selector switch, speed indicator, and reset button.

If the 8284A timer chip on your PC's motherboard is socketed, installation of the Fast88 is easy. Otherwise, to install the Fast88 you've got to clip the 8284A out and unsolder the remaining pins from the motherboard—not the easiest of tasks.

C Wizard's Programming Reference
W. David Schwaderer
Wiley Press
W. David Schwaderer is Senior Programmer at the Department of Experimental Office Systems in IBM's Research Division's San Jose Almaden Research Center. His book is not an introduction to C, but is a good reference for troubling areas of the language. Within four main sections ("C Language Overview," "C Operators," "Preprocessor and Statement Reference," "(Non) Standard Library File I/O") various topics are given short, thorough one- or two-page discussions. Appreciable effort has gone into the typesetting, headers, typefaces, tables, and charts to clarify the material. It's a useful book. □

HELP
TESTED WITH PC NETWORK
is at hand

HELP/Control™ - an on-line help subsystem for the IBM-PC. Increases the value of your software. Save development time and money.

HELP/Runtime. A few simple subroutine calls add context sensitive on-line help to your application. **HELP/Runtime** includes tested interfaces for Microsoft C, Lattice C, Turbo Pascal, IBM BASIC (Interpreter and Compiler), Microsoft FORTRAN, IBM COBOL and assembler. It is distributed with demonstration programs in each language.

HELP/Popup. Add a powerful help system to existing applications, even in dBase or 123, without reprogramming, even without a programmer. It may be memory resident, or, installed with an application, it terminates when the application exits, releasing its memory.

HELP/Generation. Use your favorite editor and our concise screen definition language to build your help files. Compile them into a help system usable by either **HELP/Runtime** or **HELP/Popup**. The package includes sources for sample help files illustrating such features as full-sized or windowed screens.

HELP/Convenience. The screens include highlighted captions. The user selects a caption with the cursor control keys and advances to a new screen, just as with 123.

HELP/Documentation. A detailed manual, both on-line and printed, for the documentation writer and programmer includes instructions which may be incorporated into the user manual.

HELP/Environment. PC-DOS 2.0 or greater is required. **HELP/Runtime** requires approximately 9K for code and buffers for full size help screens.

HELP/Pricing. The complete package (software, both manuals, and demo programs) costs $125.00 and includes a royalty-free license to add **HELP/Runtime** to your applications and to make 25 copies of **HELP/Popup**. A demonstration diskette, including the on-line manual, costs $15.00. A free update to Release 1.1 is available to registered owners. To order, or for more information (including dealer, multiple-copy and site-license pricing) call MDS at 207/772-5436. We accept MasterCard and VISA.

MDS

MDS, INC., P.O. BOX 1237, PORTLAND, MAINE 04104

Number 69 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ Billboard is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer. The goal is to provide our readers with a wide range of information on current products and updates. PJ Billboard serves our advertisers by providing a low cost, flexible format in which they can present their products to our readers.

The standard layout is four columns, 9 picas wide, with copy set in 7 point on 8 point leading. PJ Billboard offers a 400-character product description, including your company name, address, and telephone number. Boldface and italics are available.

The listings are $65, prepaid, non-commissionable for a one-time insertion. Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the Programmer's Journal Advertising Department at 503-484-0602 for more information.

Send for Free Information
An advanced book for Turbo Pascal Programmers. NOT A BEGINNER'S TUTORIAL! In-depth, thorough, practical, technical. Chapters written by Turbo Pascal experts. Topics include: Optimization techniques, using interrupts, build a Pascal subset compiler, understand bit mapped graphics, and more.

Rockland Publishing
190 Sullivan, Ste. 109
Columbia Falls, MT 59912

UNDER-C™ LIBRARY
With IBM-PC & UNDER-C™ LIBRARY (supports Microsoft C [Version 2]; Lattice C; Instant-C; Comp. Innov. C86; DeSmet) access DOS & BIOS functions. Utility programs written in C (includes MAKE and print); screen generator; source codes; not copy protected; no royalties. Manual and 4 floppies $95. Ohio residents add 6-1/2% sales tax.

Quayle Research, Inc.
6548 Edgerton Rd.
N. Royalton, OH 44133

THE SCREEN MANAGER
A complete screen management system for the PC. Up to 15 simultaneous windows. Complete control over text and attributes of every character in every window. Much more.

Comprehensive manual. Sample programs. IBM PC or true compatible.

MASM + MS C V3 + FORTRAN V3.3 or MASM + Lattice C. (Others coming). $149.95 ppd. VISA/MC. Outside USA add $10.00. NV res. add sales tax.

CONCEPT SOFTWARE
P.O. Box 26981
Las Vegas, NV 89126

FINALLY!
A Subroutine Library for Compiled BASIC
Load directories into arrays, WINDOWING, Graphics pie and bar charts, 150 more routines to add power to Microsoft QuickBASIC or IBM BASIC Compiler 2.0
30-Day Money-Back Guarantee. VISA/MC/Check/MO
FINALLY! is only $99.00 + $4.00 S&H

KOMPUTERWERK, INC., DEPT PJB
851 Parkview Blvd.
Pittsburgh, PA 15215
(412) 872-0384

Documentation
- Writing, editing, formatting for photocopying or typesetting
- MS DOS, Apple, AT&T 3B series UNIX
- Over thirty manuals
- Credits include
  Dave Winfield's BATTER UP!
  (Billboard Top Twenty)
  Action Inn Management
  (AT&T 3B2)
  SmartCAM
  (MS DOS, Apple)
  Programmer's Journal
  (Managing Editor)

Erin O'Connor
c/o Programmer's Journal
PO. Box 30160
Eugene, OR 97403
(503) 484-2162

PUT IN A GOOD WORD
When you contact these advertisers for information, please be sure to mention that you saw their product advertised in **Programmer's Journal**. Thanks!

ADVERTISER INDEX

Number	Advertiser Name	Page Number
53	Allen, Emerson & Franklin	54
30	Austin Code Works	60
84	Beagle Bros.	35
2	Blaise Computing	2
3	Borland International	5
38	C-Ware	34
64	Chalcedony Software	10
56	CompuView Products, Inc.	23
*	Creative Programming Consultants	47
83	Crescent Software	61
66	Custom Software Systems	28
5	Dair Computer Systems	29
41	Design Solutions	28
67	E-Heart Engineering	41
8	Ecosoft, Inc.	C4
11	Gimpel Software	33
42	Harvard Softworks	22
57	Hawkeye Grafix	31
58	Interface Group	1
59	L.U.C.I.D.	27
63	Laboratory Microsystems, Inc.	19
44	Lahey Computer Systems	26
69	MDS, Inc.	62
13	Micromotion	32
65	Mnematics Videotex	54
16	Opt-Tech Data Processing	30
6	Periscope Company, The	13
26	Polytron	59
82	PS Business Systems	26
31	Quantumn Information Systems	60
46	Quilt Computing	29
20	Rational Systems	6
47	Solution Systems	7
72	Speedware	55
85	Sunny Hill Software	51
35	Turbo Source Search	31
73	Whitewater Group	C2

* These Advertisers would prefer to be contacted directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ERRATA

Turbo Inline Code

James Smith reports that using MASM's .LST output to generate Turbo inline code (Issue 4.6) doesn't work properly.

"For a year," Smith writes, "I've used the output of Debug's dissassembler to generate inline statements. After writing or optimizing code, I would assemble it with MASM and then disassemble it with Debug, redirecting Debug's output to a textfile. Then I would use the utility NATIVE.PAS to convert Debug's captured output to Turbo inline code. NATIVE.PAS would output the object code as Turbo inline code, followed by Debug's disassembly opcodes in curly braces for documentation.

"In the 11th hour, someone suggested that the .LST output of MASM could be used as a source of the disassembled listing instead, because then your .ASM file comments could show up in the inline code as well. Those comments could provide a guide to your assembly code references to the Turbo data segment. Alas, it turns out, upon closer examination, that in .LST, integers are in byte-normalized form instead of byte-reversed form. Somehow that slipped by us all, and I apologize humbly for my unworthy contribution to your fine magazine.

"I've since tried combining Debug's disassembly with the .LST file, taking code from one and comments from the other, but enough syntactic differences appear between Debug and MASM to make that difficult. So I've gone back to the original approach—using Debug's output. It's still useful to generate a .LST output and refer to it with a split-screen word processor when tidying up the inline code.

As Gerald Weinberg said, 'Better is the enemy of good enough.'"

Following is the original NATIVE.PAS. You name the source and target files, in that order, on the command line when you start NATIVE.

The listing is also on the Mnematics *Programmer's Journal* SIG.

```
program native;
var
  source,target: text;
  trash: string[10];
  code: string[14];
  comment: string[45];
  temp: string[28];
  i: integer;

begin
  if paramcount=2 then
  begin
    assign(source,paramstr(1));
    assign(target,paramstr(2));
    reset(source);
    rewrite(target);
    writeln(target);
    writeln(target,'inline(');
    while not eof(source) do
      begin
        code:='';
        while (code='') and (not eof(source)) do
                              readln(source,trash,code,comment);
        while code[length(code)] in [' ',chr(9)] do
              delete(code,length(code),1);
        temp:=code;
        i:=3;
        while i<length(temp) do
          begin
            insert('/$',temp,i);
            i:=i+4;
          end;
        temp:=temp + '/';
        for i:=length(temp) to 28 do temp:=temp+' ';
        for i:=1 to length(comment) do
            if comment[i]=chr(9) then comment[i]:=' ';
        for i:=length(comment) to 40 do comment:=comment + ' ';
        writeln(target,'$',temp,'{',comment,'}');
      end;  {while not eof}
      writeln(target,');');
      writeln(target);
    close(source); close(target);
  end;  {if paramcount=2}
end.
```

DOS

This one wasn't in the magazine, but is noted in DOS. Have you noticed that it is possible to create a file with an invalid file name? We watched someone use a name with a space in it. DOS applied the name to the file, would show it in the directory, but the file could never be accessed by the name. □

Peter Feldmann

We misspelled Peter Feldmann's name (consistently!) in the last issue. Our apologies to Peter.

Send Articles

We welcome article submissions, including unsolicited manuscripts, from readers. Our editor welcomes calls from readers with article ideas. Call or write for *PJ*'s author guidelines (they're pretty simple):

Editorial Department
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

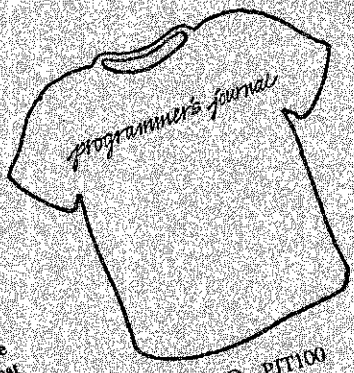
(503) 484-2162
Mnematics Mailbox 6602
Compuserve 76676,427

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

BORN TO PROGRAM PJT101

Show your true colors when cruisin' the local computer store. Let 'em know that programmers rule in an authentic black T-shirt with this hot multi-color design.



PJ LOGO PJT100

For those of you who care enough to wear the very best, we offer our world famous PJ logo printed on the T-Shirt color of your choice; White, LL Blue, Beige or Gray.

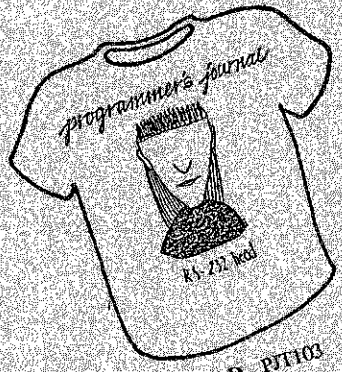
PARALLEL PORTS PJT102

These twin vessels are guaranteed to enhance communications with your favorite printer or anyone else you dump your text to. Multi-color imprint on a wine-colored T-Shirt.

NOW! SOFT WEAR FROM PROGRAMMER'S JOURNAL!
Only $9.95 each, postage paid.

By popular demand, your favorite computer jokes and cover graphics by Julia King and Ralph Bentley come alive in dynamic multi-color imprints on high quality 50/50 cotton T-Shirts. Use the form on this page to order your own unique, custom soft wear. Fill out and mail today with your payment to:

Programmer's Journal
29 W. 29th Ave., Suite 5
Eugene, OR 97405



RS-232 HEAD PJT103

Here's a connection you can't refuse. Plug your head into this neat-as-a-pin computer gray T-shirt with colorful high-tech imprint.

COMMON C BUG PJT104

Everyone's got one! Now proudly display your own specimen. The bug is brown (naturally), crawling across a gold nameplate on a beige T-shirt.

SIZES AVAILABLE: S, M, L, XL

DESIGN NO.	QTY	COLOR	SIZE	AMOUNT
TOTAL ENCLOSED				$

SEND TO:

NAME

COMPANY

ADDRESS

CITY STATE ZIP

CHECK ENCLOSED ☐ CHARGE MY: VISA☐ MC☐

CARD NO. EXP. DATE

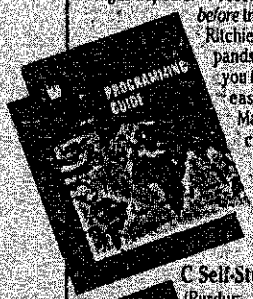
You will find a convenient order form in the front of this magazine

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

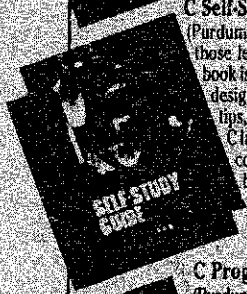
DR. JACK PURDUM

Get the proven product from the man who wrote the books on "C".

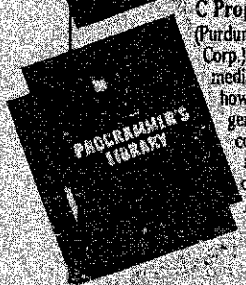
C Programming Guide
After reading the 1st edition, Jerry Pournelle (BYTE Magazine) said: "I recommend this book... Read it *before* trying to tackle Kernighan and Ritchie." The second edition expands this best seller and walks you through the C language in an easy-to-understand manner. Many of the error messages include references to this book making it a perfect companion to Eco-C88 for those just starting out with C.
$20



C Self-Study Guide
(Purdum, Que Corp.) Designed for those learning C on their own. The book is filled with questions-answers designed to illustrate many of the tips, traps, and techniques of the C language. Although written to complement the Guide, it may be used with any introductory text on C.
$17



C Programmer's Library
(Purdum, Leslie, Stegemoller, Que Corp.) This best seller is an intermediate text designed to teach you how to write library functions in a generalized fashion. The book covers many advanced C topics and contains many useful additions to your library including a complete ISAM file handler.
$22



CED Program Editor
CED now supports on-line function help. If you've forgotten how to use a standard library function, just type in the name of the function and CED gives you a brief summary, including function arguments. CED is a full screen editor with auto-flagging of source code errors, multiple windows, macros, and is fully configurable to suit your needs. You can edit, compile, link, and execute DOS commands from within the editor. Perfect for use with Eco-C88. For IBM PC, AT and look-alikes.
$29.95

C Source for Standard Library
Contains all of the source code for the library functions that are distributed with Eco-C88, excluding the transcendentals and functions written in assembler.
$10 ($20 if not with order)

Developer's Library
Contains all the source code for the standard library, including transcendental and assembler functions. Available with compiler purchase only.
$25 ($50 if not with order)

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

Number 8 on the Reader Service Card

THE FIRST PROFESSIONAL "C" COMPILER FOR UNDER $60.00

Limited Time Offer

CED TEXT EDITOR WITH "BUILT-IN" FUNCTION HELP

NOTHING IN THIS PRICE RANGE EVEN COMES CLOSE.

"Ecosoft's Eco-C is going to turn some major heads."

"Eco-C is the first compiler reviewed that has clearly begun implementing the coming ANSI standard. Eco-C supports prototyping."

"Eco-C performed well on all the benchmarks, generating code that was quite comparable to that of compilers 10 times as costly."

from:
Christopher Skelly
Computer Language,
Feb., 1986

"The driver program is another strength: it compiles and links a list of files and provides a simple MAKE capability."

"This compiler does handle syntax errors much better than average—no avalanche of spurious messages here."

from:
William Hunt
PC Tech Journal,
Jan., 1986

"Eco-C88 is a high-quality package... comparable to systems costing much more."

"Eco-C88 is one of the fastest..."

"It is convenient to use, works well, and produces acceptably compact and fast programs."

from:
Dr. David Clark
Byte, Jan., 1986

Minimum System Requirements:
To use Eco-C88 Release 3.0, you must have:
1. An IBM PC, XT, or AT-compatible computer with monitor.
2. 256K or more memory.
3. Two 360K disk drives, or a hard disk.
4. PC or MSDOS 2.1 or later to include the MSDOS linker.

Eco-C88, mini-make, memfiles and CED are trademarks of Ecosoft Inc.
IBM is a trademark of International Business Machines.
UNIX is a trademark of Bell Labs.
MSDOS and MASM are trademarks of Microsoft.

Full-featured C compiler. Supports all C features, data types (except bit fields), and operators.

New Language Enhancements. You also get prototyping, enum and void data types, plus structure passing and assignment.

Tiered Error Checking. All syntax errors are automatically flagged, but you can select the level of "lint-like" semantic error checking you want.

Complete Standard Library. Over 200 functions, many of which are System V compatible for greater source code portability.

Screen and Memory Functions. Now you can write programs that use color, cursor addressing, even ones that let you design your own graphics functions. You also get memfiles™ that allow you to access memory outside the normal data segment as a file.

8087 and 80287 Support. If you have one of these math chips, your programs will take immediate advantage of it. If you don't have one, the code automatically switches to software floating point.

Full Screen Editor. The CED editor is a full screen editor with multiple windows, macro commands, on-line function help, plus a full set of editing commands. (Requires a true IBM PC compatible.) You can edit, compile, link, and execute programs from the editor which greatly reduces development time.

Includes a cc and mini-make™. The UNIX-like cc makes compiling programs a snap. You can run cc from within the CED editor.

ASM or OBJ Output. You can select assembler or relocatable output from the compiler. Both are MASM compatible and ready for use with the MSDOS linker to produce EXE files.

	Eco-C88	Lattice	Computer Inn. C86	Microsoft	Mark Williams
sieve	12	11	13	11	12
lib	43	58	46	109	--
decef	14	13	--	10	11
matrix	22	29	27	28	29

Computer Language, Feb., 1985, p. 79. Reproduced with permission.

Orders only: 1-800-952-0472

ORDER FORM CUT & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46

- ☐ C Compiler: $59.95 _____
- ☐ C Programming Guide $20.00 _____
- ☐ C Self-Study Guide $17.00 _____
- ☐ C Programmer's Library $22.00 _____
- ☐ CED Program Editor $29.95 _____
- ☐ C Source for Standard Library $10.00 ($20.00 if not with order) _____
- ☐ Developer's Library $25.00 ($50.00 if not with order) _____

SHIPPING $4.00 _____
TOTAL (IND. RES. ADD 5% TAX) _____
PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHE
CARD # _____ EXPIR. DATE _____
NAME _____
ADDRESS _____
CITY _____ STATE _____
ZIP _____ PHONE _____

ECOSOFT

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED